Master Thesis

Consistent Inconsistency Management: a Concern-Driven Approach

A method to support the software architect in managing inconsistency in the architecture

May, 2016

Utrecht University

Department of Information and Computing Sciences
Master in Business Informatics

Author:
Jasper Schenkhuizen
Student number: 3689042

First supervisor:
dr.ir. Jan Martijn van der Werf

Second supervisor:
dr. Slinger Jansen

External supervisor:
Lambert Caljouw

Version: 1.0

Abstract

Inconsistency in software architecture is prevalent and arises inevitably due to the fact that software architecture is concerned with heterogeneous, multi-actor, multi-view and multi-model activities. Inconsistencies that remain undiscovered can be problematic.

The software engineering research community has proposed different techniques and methods to support inconsistency management, especially focusing on model inconsistency. Traditional techniques such as logic-based approaches and model checking approaches have limited applicability in day to day practice due the presence of heterogeneous and informal models in software architecture, and the lack of documentation of design decisions and architectural knowledge. The earlier inconsistency is identified, the more cost-effective it is. That emphasizes the need for a method that identifies inconsistency in software architecture in an early stage, i.e. in the form of conflicting concerns and design decisions, as these give rise to inconsistency in software architecture.

The aim of this study is to provide software architects with a sound and evaluated method that aids the architect with inconsistency management. This is done by answering the main research question: How can a software architect be supported in a structured way, to assist with inconsistency management in software architecture?

Using a rigorous process, which includes the Method Association Approach, a literature study, and various in-depth expert interviews, the "Concern-Driven Inconsistency Management" (CDIM) method is developed. Central in the method is a workshop-approach involving stakeholders that enables the architect to assess conflicts between concerns and associated design decisions, in order to detect and manage inconsistency as early as possible.

Two lightweight case studies and two in-depth interviews have demonstrated that the CDIM method is a flexible and applicable method, that can be easily tailored towards the inconsistency management needs of a software architect. Evaluation suggests that the CDIM method provides an effective way to explicate and structure the sometimes ad-hoc and implicit process of inconsistency management that software architects follow. The method also appears to be a good instrument for tradeoff analysis, guiding the architecture definition process.

Concluding, it can be stated that the CDIM method can be used to support the software architect in a structured way with inconsistency management in software architecture, especially regarding intangible inconsistency in the form of conflicting design decisions and concerns. Nonetheless, the effectiveness of CDIM for tangible inconsistency could not be established.

Preface

This master thesis is the result of roughly ten months of intensive research within the domain of software architecture. Software architecture can be seen as the linking pin between business and (information) technology. It is this role of software architecture that really has piqued my interest lately, hence it is a central element in this thesis. Although the topic of this thesis lies miles apart from the original topic that I intended to investigate, I never left the software architecture domain. My internship at Unit4 was a burdensome and turbulent period of eight months. The acquisition of Unit4 by an investment company hampered my progress and presented various challenges, of which one is the fact that I have had three supervisors in this relatively short period of time. As times were fierce, each supervisor clearly had different priorities and interests. After my first supervisor was outsourced – without informing me in the first place, a second supervisor – a London-based independent consultant – was assigned to me, also without my knowledge. In November, it was decided that my second supervisor was not needed anymore within Unit4, and once again, I was left to my fate. Fortunately, his successor and also my third supervisor, Lambert Caljouw, was reliable, knowledgeable and provided excellent guidance. The first days of November meant another pivot in my topic, which eventually became inconsistency management in software architecture. As I stated earlier, this thesis' current topic drifted miles apart from the original topic. In the past year, I learned that that is the rule rather than the exception. Every time Jan Martijn and I decided to overthrow the entire project by pivoting 90 degrees he mentioned: "That, Jasper, is research". This quote presents me with a nice opportunity to express my gratitude to Jan Martijn van der Werf for guiding me during the past year. His endless enthusiasm, patience, effort, and belief indicate his qualities as a great coach and thesis supervisor. Of course, I would like to thank Slinger Jansen for being one of the coolest, most relaxed, lecturers I know. A special thanks goes out to my girlfriend, Isolde, who accepted my grumpiness when things went awful, but also supported and loved me unconditionally during this period. Furthermore, I would like to thank my friends, who each and all accepted the fact that they barely saw me during some parts of the project.

Lastly, I would like to state that I think research is persistence, persistence, and persistence. And in the middle of all this persistence, flexibility should prevail. It is the combination of flexibility and persistence that characterizes high quality research, which is something that this thesis hopefully presents. All that remains to be said is: enjoy reading!

All the best, Jasper

Communication

Jasper Schenkhuizen, BSc
Student
Student number: 3689042

j.schenkhuizen@students.uu.nl

Utrecht University
Department of Information and Computing Sciences
Buys Ballot Building
Princetonplein 5, De Uithof
3584 CC, Utrecht

dr.ir. Jan Martijn van der Werf
First supervisor

j.m.e.m.vanderwerf@uu.nl

Utrecht University
Department of Information and Computing Sciences
Buys Ballot Building
Princetonplein 5, De Uithof
3584 CC, Utrecht

dr. Slinger Jansen
Second supervisor

slinger.jansen@uu.nl

Utrecht University
Department of Information and Computing Sciences
Buys Ballot Building
Princetonplein 5, De Uithof
3584 CC, Utrecht

Lambert Caljouw
External supervisor

lambert.caljouw@unit4.com

Unit4 IT Solutions B.V.
Business Software Holding
Papendorpseweg 100
3528 BJ, Utrecht

Contents

Abbreviations

ADD	Architectural Design Decisions
ADO	Adoption of Tool
AE	Architecture Evaluation
APR	Approach
ARA	Architecture Rationale
ASR	Architecturally Significant Requirement
CDIM	Concern-Driven Inconsistency Management
CON	Concerns
DSRM	Design Science Research Methodology
ICA	Inconsistency Causes
ICH	Inconsistency Handling
ICL	Inconsistency Classification
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IM	Inconsistency Management
IMA	Process of Inconsistency Management
IMO	Inconsistency Monitoring
ISO	International Organization for Standardization
ITY	Inconsistency Types
MAA	Method Association Approach
NFR	Non-functional Requirement
OPP	Opportunities within Inconsistency Management
PDD	Process-Deliverable Diagram
PRIO	Prioritizing Concerns
QA	Quality Attribute
RSA	Role of the Software Architect
SA	Software Architect - or Software Architecture
SCP	Scoping

List of Tables

1 Introduction

The conventional concept of inconsistency is easy to understand. As Finkelstein [47] amusingly puts it: it is simply stating two contradictory things at two different moments in place or time, such as asserting that both *the sky is blue* and that *the sky is not blue*. Of course, inconsistency may not directly be a problem, until you try to perform an action in the real world upon the inconsistent assumptions - selecting a coat for instance. Depending on the situation, the resulting actions could interfere each other: *putting on a raincoat* and *not putting on a raincoat*.

> "*Inconsistency has many causes. Foremost among these is that it results from collaboration of multiple actors, each with different opinions, views and interpretations on the real world.*" [47, p. 1]

It is the collaboration of multiple actors, working together on a various range of products, that characterizes the domain of software architecture (SA). Software architecture deals with the complexity in current information rich systems [40, 103], making it a convoluted discipline. Complexity in software architecture emerges due to many factors [95]: the needs of various system stakeholders; social and political contextual factors; constraints regarding the development, maintenance, implementation and operation of the system; and the required characteristics of the system itself. Together, these constraints and interests form the system's stakeholder concerns. The fact that software architecture and software engineering are involved with different actors, stakeholders, concerns, views and activities inherently results in inconsistency [47, 113, 116]. Finkelstein [47] argues that a major cause of inconsistency is the fact that multiple actors need to work together, each with different views, opinions and interpretations. For instance, different stakeholders involved in architecture have different languages or development strategies, address different stages of the development, have partly overlapping or non-overlapping areas of concern, and have different technical or economic objectives [113]. In addition, a lot of architectural knowledge is contained in the heads of involved architects and developers [93]; and documentation of architectural design decisions (ADDs) [159] and architecture design is often not maintained over time [77, 128]. Architectural information that is available, can be specified in a wide range of formats such as ad-hoc informal drawings, more formal models, or semi-structured text documents. Potential interference between different assertions is referred to as inconsistency, and inconsistency in software architectures is found to be prevalent [53, 129].

Inconsistency in those assertions can have negative consequences [72, 108, 116], to the extent that inconsistency can result in the need for refactoring, cause delay during development, or pose threats to a system's quality attributes such as the reliability or the performance: two architects that by accident design interfering services that together introduce an extreme load on the server, can reduce a system's performance severely.

Luckily, inconsistency is not necessarily a bad thing [111]. Spanoudakis [139] explains this by arguing that inconsistencies can reflect conflicts between different perceptions or goals of the stakeholders involved, indicate aspects of the system that deserve more attention, or promote the inspection of design alternatives. These upsides of inconsistency are known to the software engineering community, and supported by empirical studies, such as [112] or [14]. However, it has to be noted that these benefits can only be fruitfully adopted if inconsistencies are tolerated for some time, or used 'as drivers of managed interactions among the stakeholders that can deliver the above benefits' [112, 139, p. 2].

The fact that inconsistencies may have both negative and positive consequences, indicates that maintaining *consistency* at all times is not the best solution; positive implications of inconsistency might not be achieved, and a lot of valuable effort might be spent on mitigating inconsistency. Rather, inconsistency should be *managed*, that is, identifying inconsistency, leaving inconsistency where it is acceptable or desirable, and deferring or solving inconsistency when it is needed [47,116].

1.1 Problem Statement

A lot of architectural knowledge is contained in the heads of the architects and architectural documentation is often not maintained over time. Architectural elements such as models and views vary in complexity, and stakeholders and their concerns differ. In addition, inconsistency emerges in a variety of sources such as design decisions or assumptions [34], requirements [143], code [80], or models [44], where models also have different levels of abstraction and heterogeneity. This hampers the effective management of inconsistency in software architecture.

The vast amount of related research demonstrates that inconsistency management is of interest for a long time already [49, 115]. Many authors have aimed to detect and manage inconsistency in all sorts of semi-formal or formal models [44, 72], but inconsistency in more informal models or in undocumented design decisions has not received a lot of attention. Various approaches for inconsistency management use logic-based approaches and model checking approaches to focus especially on formal models and diagrams. Logic-based approaches use formal inference techniques to detect model inconsistency, making them difficult to scale. Model checking approaches dispose model verification algorithms that are sufficiently suited to detect specific inconsistencies but do not fit well to other kinds of inconsistency [13]. Further, those approaches often require significant efforts in the form of formalization of models, prior to detecting inconsistency. In addition, the majority of the approaches use rules to identify conflicts. While rules enable efficient identification of inconsistency in models [116], establishing those rules requires a lot of effort (constructing and maintaining rulesets) before inconsistency can be managed.

In this sense, we see a gap in current research on inconsistency management, to the extent that several of the aforementioned approaches are not applicable if inconsistency is present in heterogeneous or informal models, or in design decisions living in the heads of the architects. While benefits of inconsistency management have been demonstrated in various studies [116, 139],

many approaches focus on model inconsistency, and to our best knowledge, there is no appropriate approach or method that is capable of assisting the software architect managing a broad class of inconsistencies that are present in software architecture [64]. The aforementioned argumentation leads to the following problem statement:

> *In software architecture, inconsistency management is difficult due to (1) the lack of documentation, (2) the degree of heterogeneity in models, and (3) the amount of different stakeholders, each with their own perspectives and perceptions. Related approaches are not applicable, since they focus on formal model inconsistency and require construction of large rulesets. This emphasizes the need for a structured inconsistency management approach that focuses on these difficulties: (1) inconsistency in informal models and (2) undocumented inconsistency in the form of conflicts. Currently, there is no approach that helps software architects with managing these forms of inconsistency. Software architects can benefit from such an approach, as it helps mitigating harmful inconsistency, and helps embracing desirable effects of inconsistency.*

1.2 Research Focus

By having a structured approach towards inconsistency management, for example in the form of a method, the architect can possibly exploit some of the positive effects of inconsistency, and create the interaction among stakeholders that delivers the aforementioned benefits [112]. Furthermore, if architects are able to carefully manage and resolve certain inconsistencies, it allows them to possibly mitigate the negative consequences of inconsistency. In addition, inconsistency is easier to resolve if it is detected at an early stage [64]. The 'first' elements that a software architect has to consider are *concerns*. "*A concern about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture*" [130]. Overlapping or conflicting objectives and concerns are a source of inconsistency [113], and concerns persist throughout a project, as opposed to models or documents, that can become unreliable over time [137]. Managing inconsistency on the basis of concerns is therefore an area worthy of study. If an architect is able to apply such a method, it can increase insight in the architecture and promote stakeholder discussions. This leads to the objective of this thesis:

> "*Our objective is to develop and evaluate the Concern-Driven Inconsistency Management (CDIM) method that supports the software architect in a structured way with the discovery and management of inconsistency in the software architecture.*"

Specifically, within this context, the objectives of this research are to:

1. Identify the aspects that determine the shape of the CDIM method, such as the role of the architect and the main software architecture concepts.
2. Examine literature within the domain of inconsistency management and assess related inconsistency management techniques that can form inspiration for the CDIM method.
3. Gather the requirements that drive the development of the CDIM method, both from practice as well as from related literature, such as architectural evaluation methods.
4. Develop and design the activities and the deliverables of the CDIM method, preferably based upon validated relevant methods.

5. Evaluate the CDIM method thoroughly, by instantiating it in practice and gauging the usefulness and intention to use.

The proposed CDIM method (a) identifies concerns of different stakeholders, as they are a source of inconsistency [113], and (b) provides a handle for discussing overlapping concerns in order to find and manage inconsistency, as overlap is a precondition for inconsistency [49]. The method consists of a 7-step cyclic process based on the Plan Do Check Act (PDCA) cycle [79], work of Nuseibeh [116], Spanoudakis [139], and related architectural analysis and evaluation methods.

1.3 Contributions

The research in this thesis adds value in a variety of ways. First, the software architecture research community is helped by insights gained in this thesis, as the higher goal of software architecture research is to improve software architecture quality, which we believe is something that the CDIM can contribute to. Second, research in software architecture domain that addresses concerns [95, 129] or proposes concerns as first class entities in architecture [67] could be helped by our evaluation of using a concern-driven method in inconsistency management practice. From a practitioner's perspective, this thesis is relevant because it aims to present architects with an approach to manage inconsistency in the architecture.

Since the CDIM method is lightweight and flexible, it can be executed in relatively short amounts of time and with limited resources, in places where time and focus are limited. Not addressing those aspects could form a barrier for adoption, as time and resources can be difficult to achieve in industrial settings [155]. As opposed to more formal approaches that specialize in model inconsistency, the CDIM method is possibly less time-consuming, which is advantageous with respect to software architects. Especially in organizations where documentation is lacking or where design decisions are scarcely documented, the CDIM method might be useful.

Design Science Research Cycle In addition, this thesis presents the Design Science Research Cycle (DSRC), a research framework based on work of Hevner [65] and [151]. In an extensive analysis of design science literature, we found that the 'three cycle view' of Hevner [65] combined with the Regulative Cycle of Wieringa [151] results in a research framework that is both complete as well as practical. Appendix B discusses the DSRC in full.

1.4 Scope

As software architecture is a large and diversified research discipline, this project simply cannot address all sub topics of SA. Therefore, we primarily focus on stakeholders and their concerns, and how the role of software architecture and the architect can contribute to addressing those concerns. Furthermore, as the CDIM method will not be a formal method, previous approaches from literature are discussed relatively shallow, due to the fact that they are not related as such. There are various forms of inconsistency, and inconsistency arise in a multitude of elements. As the focus of this research is to support the software architect with inconsistency management, we do not focus on inconsistencies in low-level specifications or code, as those forms of inconsistency are especially relevant to developers. Neither do we aim to help the software architect with requirements engineering or analysis, as that requires an entire study on its own.

1.5 Thesis Outline

This thesis is organized as follows: Chapter 2 describes the research approach followed in this thesis. Chapter 3 creates the context and foundation for this project, while Chapter 4 discusses concepts in the domain of inconsistency management. Chapter 5 describes formulation of the CDIM's requirements through expert interviews and literature. The approach to develop the CDIM method is discussed in Chapter 6, followed by a description of the complete CDIM method in Chapter 7. The results of evaluating the CDIM are presented in Chapter 8 and discussed in Chapter 9. A discussion on validity and reliability is presented in Chapter 10 and the answer to the main research question is presented in Chapter 11.

2 Research Approach

This chapter describes the research approach followed in this thesis project. One main research question and six sub research questions guide the research in this project and are described first. Second, we discuss the research philosophy adopted in this project, followed by an explanation of the research process. This section ends with a discussion on what we call the Design Science Research Cycle.

2.1 Research Questions

RQ: "*How can a software architect be supported with inconsistency management in software architecture in a structured way?*"

The goal of this research is to aid a software architect (SA) in structurally managing inconsistency in a software architecture by developing and validating a method. This requires a clear understanding of architectural concepts. A software architect fulfills a certain role with respect to the architecture of a system, so discovering the role of a software architect helps to understand what a software architect needs in such a method:

SQ1: "*What is the role of the software architect?*"

Answering this research question will provide a clear conceptualization of how software architecture is designed and documented, how it helps a software architect in managing complexity [95], and how it shows whether a system satisfies the stakeholders' concerns. The notion of inconsistency has a lot of dimensions and definitions in software architecture and software engineering. Aiding a software architect in managing inconsistency requires insight in the characteristics of inconsistency management in the domain of software architecture:

SQ2: "*What identifies inconsistency and inconsistency management in software architecture?*"

This sub research question yield a clear understanding of the dimensions of inconsistency in software architecture. Various techniques to check and handle inconsistency have been developed

in the past years [52, 108]. Taking into consideration the insight from the two previous sub questions, we browse the available literature on the means and techniques for inconsistency management in software architecture:

SQ3: "*Which means and techniques are available to detect inconsistency in an architecture?*"

Using the insights from sub question 1, 2 and 3, a method is developed. The starting point for development of the method is a set of requirements, derived from interviews and related literature:

SQ4: "*What are the requirements of the method?*"

By addressing the requirements we develop a structured and generic approach that uses concerns to help the architect managing inconsistency in the architecture:

SQ5: "*What are the activities and the deliverables of the method?*"

This sub question produces the method and a conceptual tool which assist a software architect in discovering and managing inconsistencies. The last sub research question will address how to evaluate the effectiveness of the method:

SQ6: "*To what extent helps the method software architects in managing inconsistency in the software architecture?*"

The method helps the software architects when (a) the development requirements are satisfied; (b) when architects perceive the method as useful; (c) and when architects have an intention to use the method. The following 6 bullet points address how each sub research question is answered.

- **Sub question 1** will be answered by means of a literature study. Knowledge extracted from literature will form the basis for the method, will ensure the method fits in the daily practices of a software architect.
- **Sub question 2** will be answered by means of a literature study together with knowledge extracted from expert interviews. As a result, the boundaries of the method will arise. Which types of inconsistency will the method handle effectively?
- **Sub question 3** addresses existing techniques, that may be inspiration for the method. By answering this research question a clear overview of related work is obtained.
- **Sub question 4** will be answered by extracting requirements from experts and several literature sources.
- **Sub question 5** forms the foundation for the method. The sub question will be answered by using the Method Association Approach (MAA) in combination with advice from the expert interviews.
- **Sub question 6** will be answered through the use of two lightweight case studies and two expert interviews.

2.2 Research Philosophy

This research project is strongly motivated by the desire to enhance a certain situation by the development of an artifact. Design science research[1] is also motivated by this desire [136], and the

[1] from now on mostly referred to as design science

notion of an artifact is a central element in design science [118]. Therefore, the way of thinking and working in this thesis project has been chosen to fit the design science research paradigm. Our beliefs are supported by the fact that the problem is experienced in real life, and the fact that design science transcends academic- and IT disciplines [7], as does the central problem in this thesis.

The design science research paradigm originally stems from engineering, and science of the artificial [136]. It is a research paradigm that focuses on problem-solving, by seeking to create new and innovative artifacts in order to improve the environment [**?**, 65]. Hevner discusses design science in IS research in [65,66]. In the latter, he complements the insightful work of Iivari [73] and presents a three cycle view of design science research. The framework of Hevner [65] is based upon the IS research framework found in [66] and is focused on three inherent research cycles. The first, the *Relevance Cycle*, forms the connection between the situational context of the research project and the design science activities. In the center, the *Design Cycle* is found, which forms the core of the project by iteratively cycling between building and evaluating the artifacts and methods. The *Rigor Cycle* bridges the design science activities with the knowledge base of the project. The relevance cycle commences the research project with an environment that provides requirements for the research, as well as evaluation criteria. Hevner [65] stresses that the evaluation of the output of a design science research project is very important, as improving the environment was the main goal of initiating the project. Careful evaluation is thus required. The rigor cycle is characterized by providing past knowledge to the project and by adding project experiences, testing insights and developed methods to the knowledge base. "*Research rigor in design science is predicated on the researcher's skilled selection and application of the appropriate theories and methods for constructing and evaluating the artifact.*" [65, p. 3] Hevner emphasizes that it is unrealistic to insist that all design research must be grounded on descriptive theories; although theories can serve as origins of creative ideas. The design cycle is described by Simon [136] as the phase in which design alternatives are generated and evaluated against the requirements.

The three cycle framework is based upon a set of 7 guidelines for executing design science research in the IS domain. These guidelines describe the characteristics of well-designed research. In the following list, the guidelines are presented, including a motivation of how this project satisfies each objective:

1. **The main goal is the creation of an artifact, concentrated on a specific problem.** In this project, the main goal of the research is the creation of the CDIM method, addressing the problem of discovering inconsistencies in software architecture.
2. **The artifact should be relevant for addressing unsolved and important business problems.** Assisting the software architects in inconsistency management of intangible inconsistency and heterogeneous models could lead to prevention of problems in software architecture, and possibly lead to utilization of the benefits of inconsistency.
3. **The validity, benefit, quality and efficacy should be evaluated carefully.** By performing two lightweight case studies, the benefit for the software architect, the ease of use, and the intention to use are evaluated carefully.
4. **The research should serve a noticeable and relevant contribution.** The relevancy of the contribution is measured by the intention to use the artifact by the software architects, and will thus provide a metric for relevancy.
5. **The development and evaluation of the artifact should be executed rigorously.**

The CDIM method is developed rigorously on the basis of the Method Association Approach (MAA), literature study, and expert interviews. More detail is provided in Figure 2.4.

6. **During the development of the artifact the researcher is ought to use existing grounded theories and knowledge in order to formulate a suitable solution for the underlying problem.** The CDIM method is largely based upon existing candidate methods, identified and gathered in the MAA. However, Hevner [66] also states that not everything can be based on grounded theory, but that some theories serve as a basis for creativity. Some components of the CDIM method are extracted from expert interviews.
7. **The research should be effectively communicated towards the relevant research audience.** By producing a thesis and an article, the work of this research is effectively communicated towards the audience. The DSRM process structure of Peffers [120] helps with communicating this work effectively.

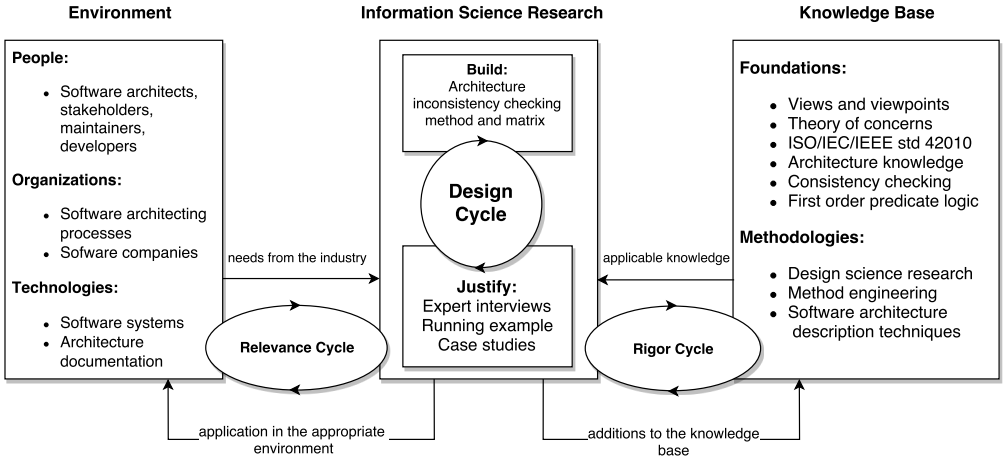


Figure 2.1: The IS research framework [66], tailored to the architecture CDIM method.

Figure 2.1 illustrates these guidelines in an IS research framework [66], adapted towards the context of this research project. The environment consists of software architects, maintainers, developers and other stakeholders that benefit from inconsistency management. Organizations where this problem can be identified are all organizations that make use of software architecture in any form. Software architecture and documentation is needed in designing large applications and is beneficial for software development [54, 121], which are technologies that could experience this problem in practice. The knowledge base of this research comprises of theories on views and viewpoints [24, 90, 129], theories on concerns [95], the ISO/IEC/IEEE standard 42010 for Architectural Description [74], and many works on inconsistency checking [16, 108]. The CDIM method is justified using two lightweight case studies. The IS research framework [66] helps with assessing the end product but does not offer a structured approach for the development and evaluation process of the CDIM method. Such a structured approach is the Design Science Research Methodology (DSRM) process, explained in the next section.

2.2.1 Design Science Research Methodology (DSRM) process

Peffers [120] provides a Design Science Research Methodology (DSRM) process existing of 6 steps that fulfill the following objectives:

- **It provides a nominal process for executing Design Science Research (DSR).** The DSRM process of Peffers [120] provides a roadmap for executing the research in this thesis. It is not the only way that DSR could be done, but it has been proven to be a good way [120]. This process also forms a way of legitimatizing the work done in this project, similar to how other empirical IS research is accepted using well understood and accepted processes.
- **It is based on prior literature concerning DS in IS and similar disciplines.** The DSRM process is based on two types of prior work: design research literature, and research on design research literature (meta-research). Design research literature contains a large number of processes that are described to produce research based designs. It contains a lot of practical needs of design researchers. The latter, research on design research literature, provides a good way of showing how to do conduct design research. The combination of those two domains formed the input for the DSRM and makes it a well founded structured suitable approach for this project.
- **It provides the researcher with a mental model or template for structuring the output of the research.** Since it is important to effectively communicate the output of the research project to the intended and relevant audience (Guideline 7, Hevner [66]), this methodology of Peffers provides a way to structure the output of the research in the 6 steps. This can be done best and comprehensively in the form of a paper or thesis, which will be done both.

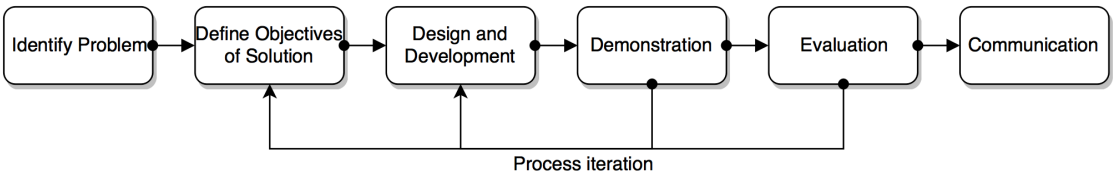


Figure 2.2: The DSRM process model adopted from [120].

The 6 steps of the Design Science Research Methodology (DSRM) process model by Peffers [120] guide a design science research practitioner from the initial goal of a project to the demonstration of a deliverable. The steps have been depicted in Figure 2.2, will be followed during this thesis project. The following sections will briefly discuss how each step is performed:

2.2.2 Identify problem

In Chapter 1 (Introduction) the problem has been formally stated. Managing inconsistencies is important [72,108], but remains a difficult and error-prone task. Earlier approaches often prescribe particular notations, or large correspondence rule sets, enforcing inconsistency. Other approaches focus heavily on documentation and models [42], which requires that architectural structures and assumptions have to be documented before they can be analyzed. In addition, inconsistency

is desirable in some cases, and maintaining inconsistency at all times proves to be counter-productive [47]. Architects need a structured approach towards inconsistency management, in which they use their own expertise, without being constrained or being forced to use certain notations.

2.2.3 Define objectives

The goal of this thesis project is to design and validate the CDIM method which uses concerns as first-class entities to aid the architect in detecting and managing inconsistencies in software architecture. The design of the CDIM method will seek to meet several sub objectives[2]. These form the **requirements** (see Fig. 2.3), distilled from three sources: literature on Architecture Description Languages (ADLs), literature on Architecture Evaluation methods and interviews with expert software architects (SAs). The requirements of the CDIM method are described in Chapter 5.

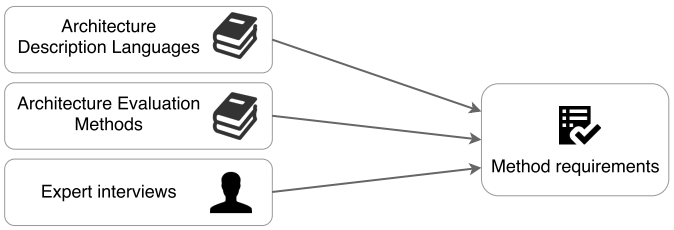


Figure 2.3: The requirements (see Chapter 5) of the method are extracted from three sources.

2.2.4 Design and development

Literature. Prerequisite knowledge on software architecture, the role of the software architect, inconsistency management, and concerns is extracted from literature via a snowballing method [152]. Snowballing better fits the needs than a Structured Literature Review (SLR) because the literature needed is dispersed across multiple domains. Furthermore, in many papers in software engineering, contextual information is not well documented, or the studies are not executed in a realistic setting [75]. Jalali [76] has observed that insufficient contextual information hinders the synthesis of of evidence of some studies, which makes a SLR in this case less appropriate. Both forward and backward snowballing techniques have been used. A selection of papers have been selected for of the aforementioned domains. The literature forms the input for the CDIM method as well as for the questions that are used in semi-structured interviews with experts. This is graphically depicted in Figure 2.4.

Expert Interviews. Semi-structured interviews are performed, that are aimed towards identifying the current inconsistency management practices, which steps of that process could be improved, and how concerns could help in improving the current practices. Furthermore, the experts are asked to provide a clear, brief description of their role as a SA. This helps in aligning

[2] from now on referred to as requirements.

the CDIM method with their daily practices. Experts are expected to coin requirements for CDIM and evaluate the 'features' (features are the starting point of the Method Association Approach, explained later).

The amount of needed experts is determined on the rule of thumb $n + 1$, until convergence is reached, or a maximum of 10 experts has been found. The sampling method that has been used is called *snowball sampling*, which is useful for sampling in a domain where access is difficult [31]. This means that after an interview, effort is spent on finding other experts through the network of the interviewee. Interviewing is a data collection technique that involves a researcher talking to at least one respondent [98]. Despite being cost inefficient and time consuming, an advantage of performing interviews is that people are familiar with answering questions. As a result, people are comfortable and familiar with this data collection method [98]. Furthermore, interviews are highly interactive and they enable the researcher to clarify questions. Interviews can result in bias dependent on the type of interview that is held. The guidelines of Cohen [31] are used for designing the interviews.

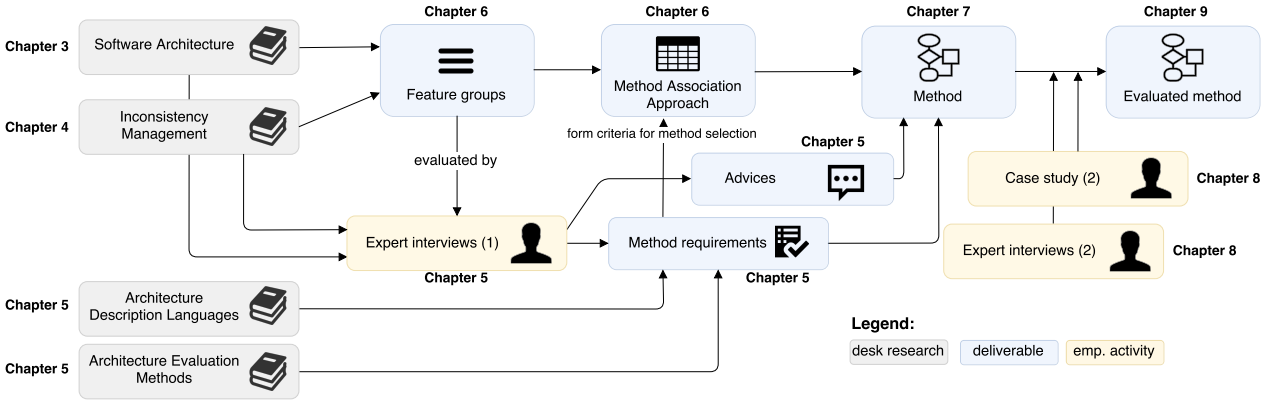


Figure 2.4: Research steps and activities followed in this research.

Method Association Approach. Knowledge from literature on software architecture and concerns, the role of the software architect, together with input from the expert interviews, will form the input for the Method Association Approach (MAA). During the MAA, a new method is developed on the basis of existing method fragments. The method fragments are assembled in a method using a 'method engineering' technique. Method engineering is a term brought to the field of information systems by Brinkkemper [21, 62, 145]. According to [21, p. 276], method engineering is the 'engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems'. The CDIM method is modeled using a technique that is called *Process-Deliverable Diagram* (PDD) [145], a technique that is especially developed for method-engineering purposes. A PDD example is depicted in Figure 2.5. A detailed description of method engineering and the method association approach is included in Chapter 6.

Figure 2.4 describes the flow of research activities and deliverables in this thesis. Each element in the figure is annotated with the respective Section (i.e. Chapter) in which it is discussed or presented.

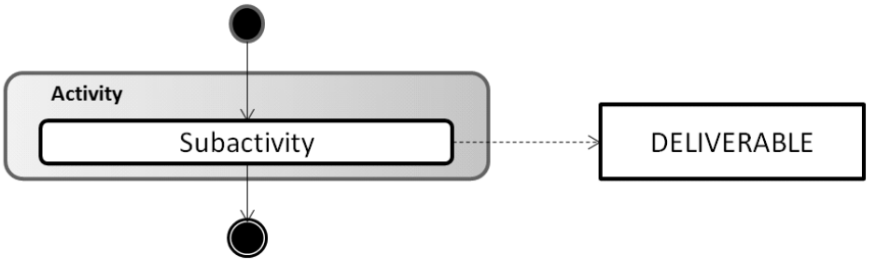


Figure 2.5: An example of a PDD-fragment, adopted from [78].

2.2.5 Demonstration

The CDIM method is partially instantiated through a conceptual tool called 'CDIM'. This tool is made using Invision in combination with Adobe Photoshop. Demonstration of the CDIM method is done through a detailed and extensive walk-through of the method, showing the conceptual screens (Appendix C), and working out examples of a deliverable of the CDIM method. After the demonstration, the CDIM method is evaluated. The next section will describe how the CDIM method will be evaluated in detail.

2.2.6 Evaluation

The purpose of the evaluation is to validate the CDIM method and to demonstrate how well the CDIM method assists the software architect in managing inconsistencies in the architecture. However, from both a practical and theoretical point of view, it is infeasible to *actually measure* whether a SA discovers more inconsistencies in a software architecture by using the CDIM method:

- It is infeasible to execute an entire instantiation of the method. Demanding such a large amount of time, employees and resources is not appropriate, considering the fact that organizations time and employees are valuable resources. Doing a small execution of the method with 2 architects and 4 other stakeholders, for 2 hours, would already require 12 man hours in total.
- Measuring the decrease in the number of inconsistencies would require two similar environments. In one environment participants would be working with the method, and the other situation would be a control environment. Requesting both groups to measure the amount of inconsistencies before and after, would yield whether the number of inconsistencies has decreased. To do this, both environments need to be perfectly the same in order to draw any valid conclusions. It is infeasible to find exactly the same companies with the same circumstances, as their will be always cultural, political and social differences. Using the same group of people (dependent samples) in two situations results in a learning effect, which biases the outcomes.

Given these challenges, analysis of decreased inconsistencies is not an applicable means to assess the effectiveness of the CDIM method.

Evaluation Criteria

As an alternative, the evaluation is aimed towards obtaining results from which improvements for the method can be extracted. Thus, the evaluation is primarily a learning activity. Four criteria have been formulated to validate the CDIM method: *requirements satisfaction*, *perceived usefulness* and *strengths* and *weaknesses*.

Requirements satisfaction This criterion refers to the extent to which the CDIM method satisfies the requirements that are elicited and presented in Chapter 5. The requirements are partly derived from the problem description, thus the extent to which the requirements are satisfied indicates the extent to which the CDIM method addresses the identified problems.

Assessing the satisfaction of the identified requirements (see Chapter 5) is done partly by presenting the participants with a set of 12 statements. Out of the 21 requirements identified in total, 10 are addressed by the statements presented to the participants. Table 8.3 in Chapter 8 describes how the assessment of the requirements is covered.

Perceived usefulness Useful is defined in the Cambridge Dictionary as "effective; helping you to do or achieve something"[3]. Usefulness is the state of being useful[4]. In this context, the usefulness of the CDIM method indicates whether the CDIM method helps the software architects with achieving their goals for this method. Assessing the *perceived* usefulness has its limitations: they are discussed in Chapter 10. By evaluating the CDIM method on perceived usefulness, it is aimed to assess the perceived usefulness of the holistic method, as well as the usefulness of a number of its important activities. Furthermore, the perceived usefulness of (1) using concerns as a basis for inconsistency detection, (2) the matrix, (3) inconsistency diagnosis phase, (4) the inconsistency handling phase, and (5) the applicability of the method in different environments are assessed.

Strengths and weaknesses The strengths of the CDIM method indicate the benefits of using and applying the method. The weaknesses indicate the areas that form obstacles in applying or adopting the method. Results of assessing the strengths and weaknesses should yield possible areas that exploit the quality of the method, and areas that hinder the effectiveness of the method. Assessing possible weaknesses of the CDIM method should yield results from which improvements can be extracted.

Evaluation method

The CDIM method will be evaluated in two lightweight case studies and in two expert interviews, using expert software architects. The next section describes the motivation for using a (lightweight) case study. Section 8.1 describes the details of the method used for evaluation. The TAM (technology acceptance model) [35] and the UTAUT (unified theory of acceptance and use of technology) [148] model have been considered as a means for evaluation, but were discarded, because the instantiation of the CDIM method is conceptual, and cannot be called an information system. The aim of TAM and UTAUT is to explain user intentions to use an information system

[3] http://dictionary.cambridge.org/dictionary/english/useful
[4] http://dictionary.cambridge.org/dictionary/english/usefulness

and subsequent usage behavior. Evaluation of the mature instantion of the CDIM method with TAM or UTAUT is a direction for future research.

Lightweight Case Study

Evaluation will be in the form of case studies, because we observe the following:

- The CDIM method is designed to be adopted in practice;
- The boundaries between the CDIM method and how it will be used by the software architect are not evident;

Case study research is 'the study of an instance in action' [2], and a case study investigates a phenomenon within a real-life context [158]. In addition, case study research is suitable when the boundaries between the phenomenon and the context are not clearly evident [158]. The two case studies will be designed using the guidelines and case study method of Yin [158], supported by Cohen et al. [31].

During the project, there are some time restrictions. Furthermore, time of available expert software architects is scarce, limiting the possibilities to instantiate the CDIM on a full scale. As a result, the case studies that are executed are limited in scope. Limited refers to a limited amount of concerns, a virtual involvement of stakeholders, and a small part of an architecture.

2.2.7 Communication

The problem and its relevance, the CDIM method, the rigor of its design, and its effectiveness are described in this thesis document. It forms the communication part of this research. The results of the evaluation will be described in Chapter 8. Chapter 10 discusses the findings and other relevant aspects of this research project. A scientific article will be published in order to contribute to the knowledge on inconsistency management in software architecture.

3

Background: Software Architecture

The aim of this chapter is to clearly present the knowledge that has been gained through an extensive analysis of literature. The knowledge forms the input for a conceptual model, created and presented at the end of this chapter. Gained knowledge also forms input for (1) research on inconsistency management (Chapter 4), (2) the questions used in the expert interviews (Chapter 5), (3) the Method Association Approach (MAA, Chapter 6) and (4) development of the CDIM method. In addition, this chapter seeks to answer the first sub research question.

3.1 Introduction

According to the IEEE an architecture is the whole of "*fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*" [74]. An architectural *element* is "a fundamental piece from which a system can be considered to be constructed" [130, p. 1]. Examples are: libraries, programming frameworks, software units, subsystems, database management systems, or services [130]. The ISO/IEC/IEEE Standard 42010 [74] distinguishes between architecture and architecture description. The architecture exists without ever being written down, and the *architecture description* is what is written down as a work product. The architecture description is sometimes referred to as the *intended/conceptual architecture*, while the architecture of a system is referred to as the *implemented architecture* [122].

The term *software architecture* is used for reasoning about individual software systems, and software architecture as a research discipline has been covered by various authors [**?**, 8, 15, 122]. Software architectures present helpful abstractions for managing complexities within a single software system [15]. A software architecture description can function as a shared conceptual model of a software system, often depicted at a high level of abstraction [70]. There is a plethora of definitions for the term, making it a blurry concept (see [122]). However, there is a general consensus that it describes the set of structures of a system, and the relations among them, which enable the architect to reason about a system [8]. An interesting definition comes from Medvidovic [105], who defines software architecture as 'the set of principal design decisions about the system'. According to him, design decisions encompass all the aspects of the system under

development, including [105]:

- design decisions related to system structure – for example, "*there should be exactly three components in the system, the 'data store', the 'business logic', and the 'user interface component'*";
- design decisions related to behavior - for example, "*data processing, storage, and visualization will be handled separately*";
- design decisions related to interaction – for example, "*communication among all system elements will occur only using event notifications*";
- design decisions related to the system's non-functional properties – for example, "*the system's dependability will be ensured by replicated processing modules*";
- design decisions related to the system's development itself – for example the process that will be used to develop and evolve the system;
- design decisions related to the system's business position - for example its relationship to other products, time-to-market, and so on.

According to Medvidovic, design decisions are 'principal' if they have a degree of importance that grants them their 'architectural status' [105]. This implies that not all design decision are architectural, at least, not all design decisions affect the architecture of a system. This is an important distinction between the process of software architecture and the process of software development. The exact boundary of a 'principal' design decision is dependent on the context of the system. This is primarily determined by the stakeholders of the system. We observe thus that architecture is strongly affected by the contextual factors such as the environment and the stakeholders of a system. This demonstrates that information systems are not created 'in a vacuum, but are created in order to meet specific needs that specific groups of people have' [130]. That is where the notion of a *stakeholder* comes into play: "*a stakeholder in a software architecture is a person, group, or entity with an interest in or concerns about the realization of the architecture*" [130].

3.2 Stakeholders

In addition to using a system, a system must be built and tested, maintained and evolved, operated and governed, and the system must be owned by some party. Each of these endeavors requires new possible roles to be involved with a software system in addition to the users [130]. Each group of involved people has a set of needs, interests or requirements that need to be satisfied by the software. The different roles that have an interest in the software are referred to as the stakeholders of a certain system. A stakeholder is referred to as a person, group or entity that has an interest in or concerns about the realization of an architecture by Rozanski [129]. Dashofy defines a stakeholder as "*a person or group that has an investment, a share, or an interest in something*" [34]. In systems engineering, software engineering, or software architecture, the target of interest will generally be a software product or system. The definitions are deliberately broad; any person or organizational representative who has a certain interest in a system will be a stakeholder. Often, a stakeholder represents a role instead of a individual person. Understanding the role of such a stakeholder is one of the principles of architecting. The ISO/IEC/IEEE standard 42010 [74] considers the following stakeholders: users of a system,

operators of a system; acquirers of a system; owners of a system; suppliers of a system; developers of a system; builders of a system; maintainers of a system. Bass et al. [8] also present a classification of different stakeholders in their book Software Architecture in Practice. Different sources present various classifications; the following list presents a synthesized overview of the work of [8, 129]. Various stakeholder groups are involved in the majority of software architecture and development practices, however not always with the same relative intensity and frequency.

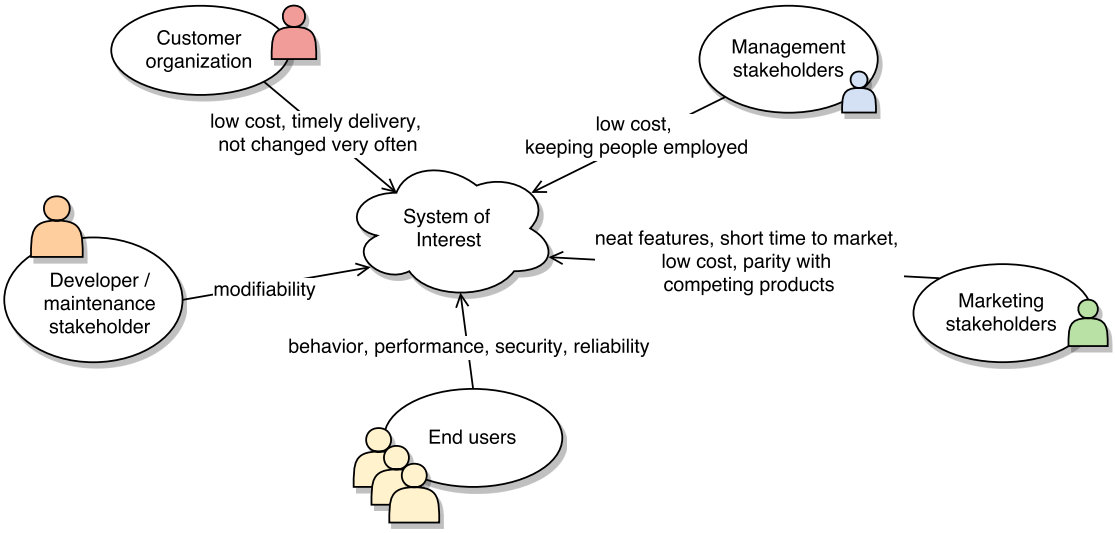


Figure 3.1: Different stakeholders and their concerns, adapted from [8].

Stakeholders are the most vital component of software architecting because stakeholders drive the "*whole shape and direction of the architecture*" [129]. They drive the architecture because the stakeholders have a stake in the system's architecture and thus may attempt to influence the architectural design decisions in various ways [5]. The architecture of a certain system is devised purely for stakeholders' needs, since stakeholders *(in)directly direct* the fundamental decisions about the aspects of a system: scope, environment, functionality, performance, scalability, and operational characteristics. Without the stakeholders of a system, there would be no system. As is shown in the conceptual model (Sect. 3.8), stakeholders are critical to a system: stakeholders all have a shared stake; namely the success of the system. However, stakeholders typically have different specific concerns and interests that they envision for the system [8]. Figure 3.1 depicts how different stakeholder groups have different concerns and aspirations for a certain system, under different circumstances. These concerns can be very diverse, and an architect should understand the nature, source and priority of the concerns as early as possible. The earlier the architect understands important concerns, the better he is able to satisfy stakeholders concerns.

3.3 Concerns

The notion of *concern* comes from the principle *separation of concerns*, which dates back to the roots of software engineering where Dijkstra coined the term 'separation of concerns' [38].

However, the development concerns of Dijkstra about design of algorithms and data structures differ from architectural and stakeholder concerns[1] about the organization of a (large) system [54]. The former has been the classical focus of software engineering, while the other type of concerns is emerged in software architecture as a significant design level that requires its own notations, theories and tools [54].

Each of a system's involved stakeholders has certain needs or requirements [95]. In addition, social and political contextual factors, constraints regarding the development, maintenance, implementation and operation of the system, and the required characteristics of the system itself result in a lot of complexity. All these factors, needs, and interests of the set of stakeholders of a system form the "stakeholders' concerns" [95]. "*A concern about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture*" [130]. Stakeholders concerns express the aspects that should be relevant to a software system in the eyes of a software architect. According to [95, p. 20] stakeholder concerns cover "*any and all the things the architect must care about in envisioning the system, meeting its requirements, overseeing its development, and certifying it for use*" [95]. Lago and colleagues argue that concerns usually result in requirements, design constraints or decisions, and that software architects must identify and manage a wide set of architectural concerns to devise a successful architecture [95]. The separation of concerns has led to the use of frameworks and architecture description approaches that make use of multiple views, where each views addresses a set of concerns [95]. Based on the definition presented above, it follows that concerns are fundamentally *conceptual*. Generally, they are not artifacts, although artifacts could represent concerns [140]. Furthermore, concerns are not low-level requirements, however, requirements represent concerns [140].

Software architects have to manage large amounts of architectural concerns in order to develop a successful architecture [67, 95]. Many concerns can be important to consider, depending on the size, scope and complexity of the system and the amount, involvement and role of the stakeholders. Figure 3.2 gives a - by no means exhaustive - example set of concerns that can be of relevance during a software project [67].

3.3.1 Classification of Concerns

A concern can be seen as a superordinate class that can refer to requirements, objectives, intentions or aspirations, which makes it a broad and vague term [130]. Therefore, we use the following classification, based on [129]: a concern is a superordinate element that could be an *architectural goal*, a *functional requirement*, or an *architectural requirement*.

Architectural Goals

Concerns are architectural goals, the objectives, the intentions or aspirations. An intention or objective both do not really need a formal definition, but can be referred to as the goal or purpose someone has, while doing certain actions. An aspiration could be seen as a strong desire or ambition that someone has. With the term architectural goals, it is aimed to describe all of them. An architectural goal typically possesses little of the characteristics of a well-formulated requirement, as it is commonly expressed using natural and imprecise language. It is unlikely to

[1] from now on, the term *concern* refers to both *architectural concern* and *stakeholder concern*, as they can be used interchangeably and refer to the same concept.

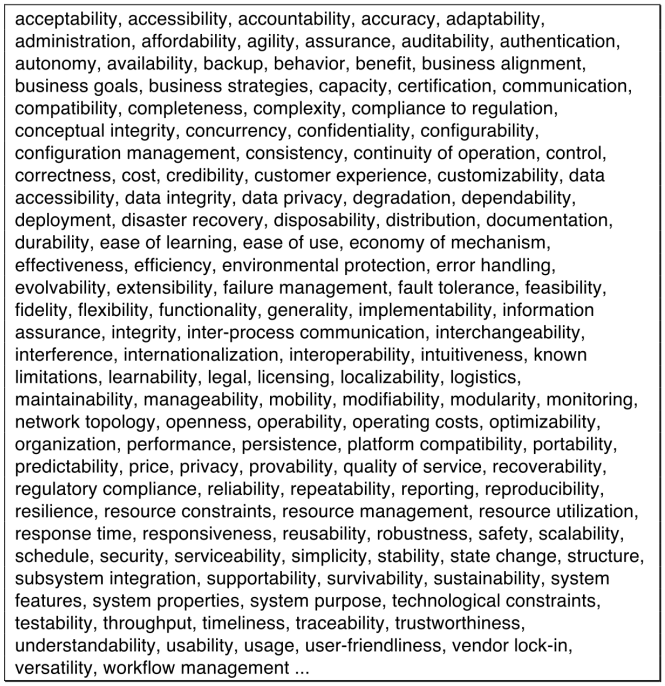


Figure 3.2: Non-exhaustive examples of common concerns from systems and software, adapted from [67].

be measurable or quantifiable, as most goals are influenced by subjective assessments, and thus there are no criteria to determine whether or not the goal has been met. Furthermore, often the aspirations or objectives have a business focus [8], which could sometimes be hard to translate into architectural solutions [129].

Constraints are also placed in this category. Constraints are design decisions with zero degrees of freedom, usually introduced by external factors such as management decisions, or insufficient resources. Examples include the requirement to use a certain web programming language, or not being able to train staff in a certain language [8]. From the perspective of an architect, these constraints could be arguable, but outside factors dictate these design outcomes.

Functional Architectural Requirements

The Software Engineering Body of Knowledge (SWEBOK) defines a requirement as "*a property that must be exhibited by something in order to solve some problem in the real world*" [17, p. 1]. Often, a distinction is made between 'functional requirements' and 'non-functional requirements' (NFRs). Lago [95] strongly opposes this distinction, as this implies that the functional requirements are half of all requirements, and the rest of the requirements can all be seen as a big pile of 'non-functional'. According to them, it is important to clearly define what comprises a NFR. Opposed to NFRs, functional requirements of a system are usually the most concrete concerns, and *functionality* is one of the best-understood concerns [95]. Often, the functional viewpoint (in view-based approaches) is the cornerstone of an architectural description. Functionality comprises

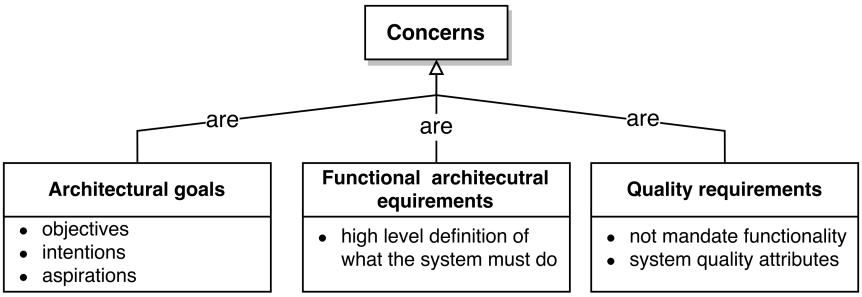


Figure 3.3: Classification and terminology of a *concern*.

the capabilities the system must provide, the services that a system performs, and the way it delivers those functionalities to the user [95]. The functionality that a system is required to deliver, is represented by the *functional requirements* of a system: a functional requirement of a system is easier to understand and more concrete than a certain aspiration a stakeholder has for the system. Functional requirements dictate what a system must do; how it must behave or how it must react to run-time stimuli [8]. Functionality does not determine architecture, given a set of different functionalities, there are various ways in which you could build an architecture to deliver that particular functionality. The topic of Requirements Engineering (RE) is outside of the scope of this thesis; the practices of detailed requirements engineering and elicitation are outside the boundaries of an architects' activities.

Quality Requirements

Quality requirements are the requirements that do not directly mandate functionality of some form. Often, this group of architectural requirements is erroneously referred to as *non-functional requirements* (NFRs) according to Lago and colleagues [95]. They argue that the term is troublesome for two reasons: first, *non-functional* implies that everything that is *not* functional, can be tied together. This suggests that once the functional requirements have been identified, the only task left is to discover and identify the NFRs; in reality the range of diverse requirements that are not functional cannot be treated equally and needs large doses of attention. Second, they argue that this group of 'architectural requirements' requires and involves a lot more than only the stated requirements; how to measure the extent to which such a requirement has been satisfied? This group of requirements is difficult to manage [129], and will be referred to as so called *quality requirements*, *quality attributes* (QAs) or *system qualities*. QAs can be seen as qualifications of the system's functions, and are measurable properties of a system that are used to indicate how well the system satisfies the needs of the stakeholders [8]. QAs qualify the functionality of a system, such as how fast a certain function must be performed, or how resilient it must be to erroneous input [8]. Functionality and QAs are independent but might as well influence each other. A functional requirement is "When the user presses the blue button, the 'help-dialog' must appear." A performance QA might prescribe how quickly the dialog should appear, or an availability QA could describe how often this function may fail or how quickly it must be repaired [8].

We conclude this section with stating that concerns form the primary drivers and building blocks for a software architecture [67]. The variety of concerns forces architects to think about the various aspects that need to be taken into account when devising an architecture, and are therefore very important. Concerns can be design constraints, quality requirements or functional requirements. Their core role leads to the vision that inconsistency management in architecture should be performed with a focus on concerns, treating them as first-class entities. The next section will present the rationale behind this vision.

3.4 Architectural Design Decisions (ADDs)

The association between design decisions and concerns is not a simple one-way relationship: concerns enforce certain architectural design decisions (ADDs). Furthermore, concerns constrain certain design decisions, thereby affecting other design decisions as well: the structure and interplay between concerns and ADDs is very complex and dynamic, and forms an own topic of study [91,93]. Kruchten identifies three major classes of design decisions: existence decisions, property decisions, and executive decisions [91], which they named ontocrises, diacrises and pericrises respectively. The ontocrisis (existence decision) has an antonym: anticrisis, referring to the non-existence of something. Ontocrises (existence decisions) state that elements will exist in the implementation. There are two types of ontocrises, namely structural and behavioral decisions: structural ontocrises result in the formation of subsystems, components in the architecture. Behavioral ontocrises are related to the satisfaction of some QAs or to how the components interact to satisfy certain functionality. This subclass is similar to *tactics* as referred to by Bass, Clements and Kazman [8]. Existence decisions are the most visible in system's design and implementation. On the other hand, *non-existence design decisions* (anticrises), are not at all traceable to a certain element in the implementation, and could be important to document. Example of such a design decision is "the system does not use Cassandra as a non-relational database management system" [91]. Subsequently, diacrises (property decisions) state overarching characteristics of the system. When they are stated positively, they can be design guidelines, and when stated negatively, these diacrises could present constraints. Properties are harder to pinpoint, and do not specifically relate to particular elements, moreover they refer to certain qualities or characteristics of the system, and are stimulated by cross-cutting concerns. From Kruchten we obtain an example of a negatively formed property decision: "*the implementation does not make use of open-source components whose license restricts closed redistribution*" [91, p.2]. Some design decisions cannot be related directly to design elements and their qualities, but are impelled by financial or managerial aspects, and affect for instance the development process and the choices of technologies, tools, people and training. According to [91] these can be referred to as pericrises (executive decisions). An example would be a decision from the management to develop the entire system in Java. Pericrises usually constrain or frame ontocrises and diacrises.

Figure 3.4 shows the range of design decisions correlated with a spectrum of possible notations. In general, the selected notation affects the level of accuracy and precision with which a design decision can be captured. Less formal languages are suitable for high-level, abstract design decisions, and more rigorous languages lend themselves for increasingly concrete design decisions. A notation like UML is somewhere in the middle: the syntax is moderately rigorous, but UML is still ambiguous to some extent. Architecture is concerned with principal [34] and early [28]

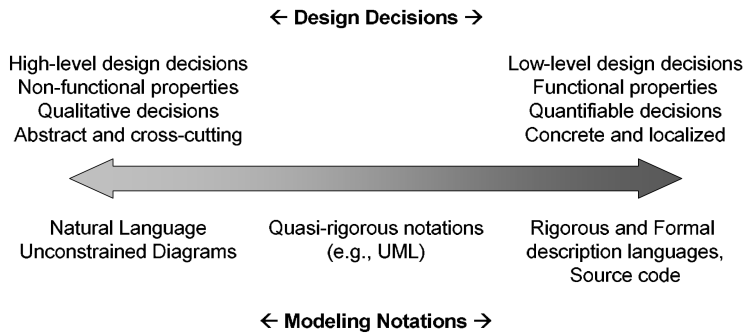


Figure 3.4: Spectrum of design decisions and appropriate notations, adopted from [34].

design decisions. We can infer that as a result, architecture is primarily concerned with the design decisions on the left side of the spectrum. This is confirmed by a growing trend towards more informal architectural description languages (ADLs), 'moving away from the technology lamppost' [105]. As architecture is defined as a set of principal design decisions, it follows that capturing those design decisions is an important activity. Design decisions are usually kept in an architecture design rationale.

3.4.1 Design Rationale

Tang, Jin and Han [142] have introduced a rationale-based architecture model that includes design rationale, design objects and their relationship. The model enables explanation of why design objects exist and on which assumptions and constraints they rest. The model is out of scope for this thesis, and so is the need to advocate the benefits of using design rationale, however their conceptual model is of interest here. They have a solid foundation of how 'design decisions' relate to 'motivational reasons', 'design outcomes', and 'architecture rationale'. According to [142], reasoning can come in two forms: *motivational reason* and *design rationale*. Motivational reasons are the goals to be achieved by the architecture, or factors that constrain the design. An example could be a requirement: however, while the requirement *itself* is not a reason for the design, the underlying *need* of a requirement is. Tang et al. [142] refer to motivational reasons as goals, assumptions, requirements, constraints or design objects. Furthermore, a motivational reason is an 'impetus to a design issue', it is a 'goal or a sub-goal to be achieved', it 'can influence a decision by ways of supporting, rejecting or constraining a decision' [142]. In other words, *a motivational reason is similar to a concern*, and forms the input for a design decision. The captured arguments and reasoning that is the result of a decision, is referred to as the *design rationale*. A decision is justified by the architecture design rationale, but the decision also creates the design rationale. The design rationale encapsulates the details of the justification [142]. It comprises of a description of the issues related to the decision, alternatives, and reasons for and against the decision. Subsequently, the result of a particular design decision is a design outcome. A design outcome is the actual consequence of making a particular decision. The graphical model of those four concepts is presented in Figure 3.5.

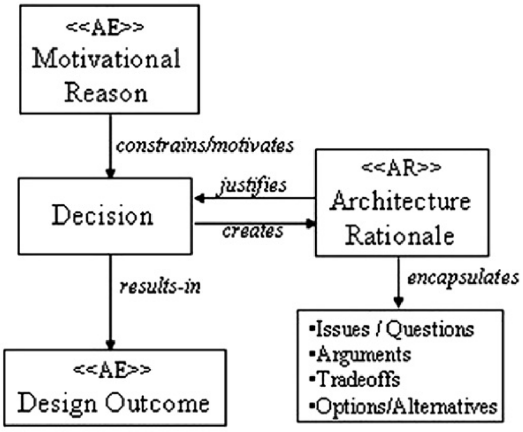


Figure 3.5: Conceptual model of design decisions, adopted from [142].

3.5 Views and Viewpoints

Understanding the wide range of concerns that stakeholders have for an architecture, and the role of stakeholders in the environment, is itself fundamental to the role of the software architect of a system. The broad set of concerns for a particular architecture introduces complexity in the process of describing a system's architecture, reasoning about design decisions, and all other steps that are involved. To address this complexity in architecture description, they are often composed of multiple views. Views are used to achieve a separation of concerns, where each view describes the architecture from the perspective of associated stakeholder concerns [74]. Rozanski and Woods define a view as "*a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders* [129]. So a view inherently addresses concerns that are framed by a coordinating viewpoint [74]. The idea of using views to describe the software architecture is certainly not new, in software development Parnas set the stage for this in 1974 [119]. Viewing an architecture as a set of design decisions, views can be defined as 'a set of design decisions related by a common concern (or set of concerns)' [34]. This definition organizes design decisions around a concern or set of concerns, and forms the foundation for the conceptual model that this chapter will deliver.

Viewpoint Frameworks. To promote reuse and speed up architecture documentation the term *viewpoint* has been introduced: "*an architecture viewpoint documents the conventions for constructing, interpreting and analyzing a particular kind of view*" [74]. Various authors have advocated predetermined sets of views and viewspoints. Appendix D discusses the important influential works. It is important to note that some authors focus on architecture documentation (e.g. [24]) while others focus on using the viewpoint frameworks for architecture design. In practice, the two activities are very much related, so the distinction here is irrelevant for this section.

3.6 Process of Architecting

Hilliard argues that the process of architecting is a life cycle practice, not a single stage or a phase [67]. This is also advocated by Poort [123], who mentions that design decisions are an architect's main deliverables. According to Hilliard, architecting is an ongoing endeavor since concerns arise throughout the life cycle. Each concernw requires new decisions, and decisions possibly introduce new concerns. As the CDIM method is intended to be used by software architect (SAs), it means that the method must fit in the daily activities of a SA. The careful analysis of the process of architecting helps to tailor and adapt the method. Architecture basically forms the linking pin between requirements and design, performing concessions to satisfy demands of both [129].

The process of architecting (referred to as 'architecture definition' by [129]) fits in between requirements analysis on one side, and software construction on the other side, yet, the boundaries are vague in practice. Requirements analysis could provide the context for architecture definition by defining the scope, desired functionality and quality properties of the system. Architecture practices on the other hand, reveal inconsistent or missing requirements, helps stakeholders understand costs and risks of their concerns. Architecture definition makes concessions between stakeholders' aspirations and what can be achieved within the limited time and budget. On the software design side, architecture presents the context for software construction. The construction of particular pieces of the architecture yield insight in problems with the current solution architecture, with the consequence that the architecture is evolving continuously during the software development life cycle [129]. Figure 3.6) shows how architecture, and especially architecture definition, connects the problem and solution space.

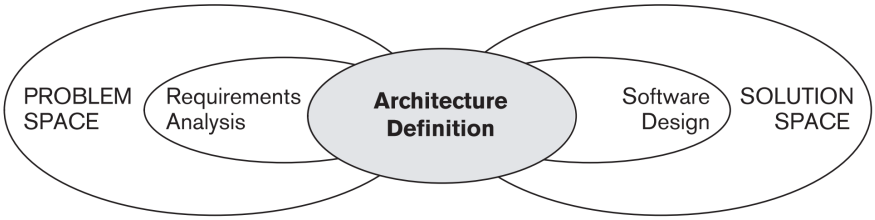


Figure 3.6: Architecture forms the connection between requirements analysis and software construction, figure adopted from [129].

A generic process of architecting is presented in [68]. In this article, the authors compare five industrial architecture design methods, and extract from the overlaps a generic architecture approach. The activities are: *Architectural Analysis*; define problems the architecture must solve and concerns the architecture must address in the form of architectural significant requirements (ASRs). An ASR is a requirement "a requirement upon a software system which influences its architecture" [117]. *Architectural Synthesis* refers to the core design step: making the design decisions. This activity moves from problem space to solution space by providing architectural solutions to architectural significant requirements. *Architectural Evaluation*: this step ensures that the design decisions solve the concerns raised. The process is shown in Figure 3.7. It is important to note that the process is happening continuously. The following three subsections will

discuss the generic process of architecting briefly. The architecture processes from Gorton [57], Bass [8] and Rozanski and Woods [129] are used to discuss each of the steps.

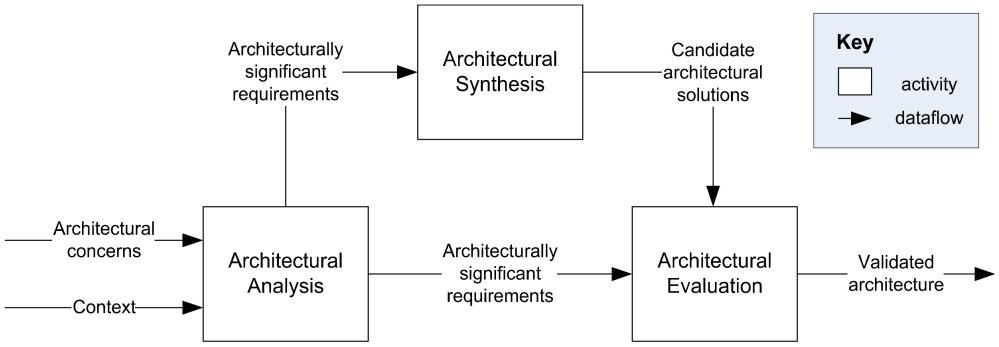


Figure 3.7: Architectural design activities, adopted from [68].

3.6.1 Architecture Analysis

Bass [8] starts the architecting process with making a business case for the system. This is a business justification to start the project. The business goals for the system determine the business case for the system. This activity is typically not performed by the SA, however, a SA should be involved in this step: estimating expected costs, benefits and risks. Architectural analysis functions as a way to characterize the problems the architecture must solve. This activity examines, identifies, and gathers the concerns in order to come up with a set of ASRs [68]. It involves identification and creation of the requirements that will drive the architecture design [57]. Bass and colleagues [8] discuss scenarios and prototypes as instruments to collect and identify important requirements. Moreover, they argue that a SA should evaluate whether the system being built is a variation on other related systems that have been constructed, since it is rare that a system is totally different from related systems. Understanding characteristics and requirements of other systems is an important part of understanding ASRs.

3.6.2 Architecture Synthesis

In this step, the architect proposes architecture solutions to a set of ASRs, moving from the problem to the solution space [68]. Architecture Synthesis is referred to by Gorton as Architecture Design [57]. Architecture design can be seen as the most difficult task a software architect undertakes [57]. Architecture design consists of choosing the architecture frameworks and allocating components. The architect chooses known candidate solutions that work, identifies relevant architectural styles [129], selects and incorporates one or more known patterns, while he orchestrates how the patterns must be integrated to form an overall architecture. The decision to include and exclude certain patterns should be dependent on the quality and functional requirements identified [57]. Bass et al. [8] state that the key to good system design is conceptual integrity. There are various strategies for designing and creating an architecture. The Attribute-Driven Design (ADD) method is an iterative method that helps the SA to choose a part of the system to design, organize all the ASRs for that part, and create and test a design for

that part. The ADD is an example of an incremental method for designing the architecture. When a general architecture framework has been devised, the next task is to define the major components that will comprise the design. In this sub step, tasks like identifying the application components, interfaces, services, and dependencies should be considered [57]. Optionally, a set of scenarios could be produced that address important characteristics and QAs that the architecture should satisfy [129]. The final step, explicitly mentioned by Bass [8], is documenting and communicating the architecture. Architectural documentation should be communicated clearly and unambiguously towards the relevant stakeholders: developers, testers, management and others. The architecture description (AD) should be informative and readable: using multiple views to document and illustrate the architecture is common [8].

3.6.3 Architecture Evaluation

During this step the architect ensures that the architectural design decisions he made are correct by evaluating the architecture [68]. Various candidate architectural solutions are measured against the ASRs. Evaluating the architecture for the qualities that it should support is crucial: many methods and techniques are available [8]. An important evaluation metric is the economic consequences of the design decisions [124]. Gorton describes two main techniques to validate the designed architecture: test scenarios and prototyping. Scenarios are associated with concerns such as QAs, and they attempt to highlight potential consequences of design decision that are encapsulated in design. There are a lot of architectural evaluation techniques that make use of scenarios [5, 39]. If the extent to which the architecture addresses the concerns and requirements is unsatisfactory, possible rework needs to be done on the architecture, based on the results of the evaluation activity [129]. The evaluation provides a context for a revision of the identified ASRs. Often this step is executed concurrently with the previous activity [129]. The output of architectural evaluation is the validated architecture [68].

The software development methodology that is followed during a project indirectly determines how often and when an architect must revisit and elaborate 'Architecture Analysis', 'Architecture Synthesis' and 'Architecture Evaluation'. Bass [8] argues that the activities should be considered in each way of working, no matter whether using a waterfall method or an iterative agile method. The following section discusses architecture and agile ways of working.

3.6.4 Agile Architecting

From the outset, architecture principles appear to confront agile values. Agile methods are often resolutely adaptive: decisions take place when changes occur, or at 'the last responsible moment'. As a result, software architecture could be perceived by agile methods as being too reliant on anticipation instead of adaption [1]. Agile methods have the drive to 'deliver value to the stakeholders' from the very first iteration. The problem is that the value of architecture is difficult to grasp because it remains invisible, whereas the cost of architecture is visible [1].

Recent research shows that architecture and agility are able to coexist well. Falessi, Cantone and Sarcia [46] discovered that software architecture is perceived as relevant by agile developers due to aspects such as evaluating design alternatives, the documentation of design assumptions and communication among members of the team. Architecture and agility can not only coexist, there are in fact multiple approaches to reconcile architecture and agile methods. Cockburn [30] suggests

a methodology for starting with a walking 'skeleton', then evolving it iteratively. When combining architecture and agility two issues are important: the amount of architecture needed, and the timeliness of architecture decisions (only a fraction of all design decisions is architectural [34,105]). The amount of architecture a project needs is dependent on the environment and context of the project [1]. The common observation is that as complexity of software grows, the relevance and criticality of software architecture grows as well. Timeliness of arhcitectural decisions is also very important in agile environments: changing them can sometimes be costly or difficult. Making design decisions too early can constrain agile development teams, and waiting too long to decide upon architectural issues can result in problems [1].

Poort states that 'the secret of making architecture work agile' is to change the view of the deliverable of architecting [123]. Similar to agile software development, an agile team does not deliver a 'big bang system', but incremental improvements to a system continuously. On the same note, agile architects should not deliver a BUFD (big up-front design) but continuously make architectural decisions. Each new decision gains more control of the uncertainties and risks surrounding complex projects. Poort [123] advices not to let agile dogmas such as "You Ain't Gonna Need It" (YAGNI) determine how much architecture should be built in. As for architectural documentation: it is not a big up-front activity that needs to be finished before everything else can be done [25]. Clements [29] dictates producing the needed documentation in prioritized stages so that the documentation evolves incrementally.

3.7 Role of the Software Architect

The title 'software architect' might be the most ambiguous and ill-defined term in software engineering domain: "*the title 'software architect' might be given to the most-senior developer or to the trusted technical advisor to managers. Or it might actually refer to the team member who creates and evolves the system's architecture*" [87]. Clements' [27] findings are consistent with this, as they identified close to 200 duties, 100 skills, and 100 knowledge areas for software architects.

So instead of viewing an 'architect' as a single job description that should fit in all circumstances, it rather should be viewed as a certain role that changes over time. In fact, each phase of a software life cycle requires different roles and skills of an architect [87]. Klein [87] presents a model that describes the architect's changing role over the life cycle of a software system (see Figure 3.8). As an 'initial designer', the architect creates structure and abstractions that can be used by the developers, and captures the essential characteristics of the system's function. Architects need to not only design the architecture, but also make sure the architecture can be implemented. The 'extender' role starts when the system is delivered. As an 'extender', the architect trades off conceptual integrity with value creation, possibly through extending it or integrating it with other systems [87]. Extending or integrating the system decreases the initial conceptual integrity, so an architect should make compromises. Architects in this phase need a solid understanding of the system as-built, including undocumented side effects. As a 'sustainer', a role that commences after the system has been in production for a while, the architect should communicate and demonstrate the system's usefulness and relevancy as the environment changes. They must understand the business value of the system and the evolving technology environment that the system resides in. All three roles are totally different, and can be very well executed by totally different members of the team. For example, Klein [87] argues: "*in many organizations,*

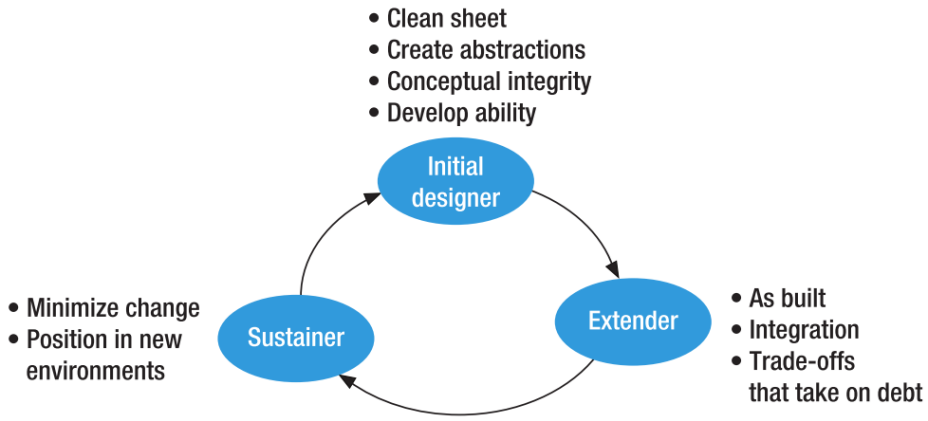


Figure 3.8: Roles of an architect during software life cycle, adopted from [87].

the initial designer is reassigned after delivering the system's first versions, and one or more developers take over as extenders."

Philippe Kruchten [92] presents a list of responsibilities of a software architect, based upon his own experience:

- Defining the architecture of the system. Understanding requirements, qualities, extracting ASRs, making decisions, synthesizing a solution, exploring alternatives, validating them, etc.
- Maintenance of the architectural integrity of the system. Doing reviews, writing guidelines, and presenting the architecture to various parties, at different levels of abstraction and technical depth.
- Assessment of technical risks.
- Working out risk mitigation strategies/approaches.
- Participating in project planning.
- Proposing order and content of development iterations. For many effort estimation aspects, or for the partition of work across multiples team, managers need the assistance of architects.
- Consulting design and development teams. Because of their technical expertise, architects are involved into problem-solving and fire-fighting activities that are beyond solving strictly architectural issues.
- Architects assist product marketing and future product definitions. The architects have insights into what is feasible, doable, or science fiction and their presence in a product definition or marketing team can be effective.

This corresponds with the list that from the work Rozanski and Woods [129], who also present a comprehensive list of architectural responsibilities. Furthermore, the list is consistent with Bredemeyer's 'Architect Competency Framework' [19], containing five pillars: technology, consulting, strategy, organizational politics, and leadership. Kruchten suggests a simple time division model for architects, which is presented in Figure 3.9. He recommends architects to allocate their time in a 50-25-25 ratio with their tasks being as follows [92]:

- Internal focus (50% of the time) is on architecting per se: designing the architecture,

prototyping, evaluating, documenting.

- External focus (50% of the time) can be divided in two aspects:
  - Inwards (25%) refers to getting input from the environment: talking to customers, product managers, developers and other stakeholders. Furthermore, the architect should learn about technological developments, related systems, and architectural practices.
  - Outwards (25%) refers to providing actual information to other stakeholders, communication and definition of the architecture, project management, etc.

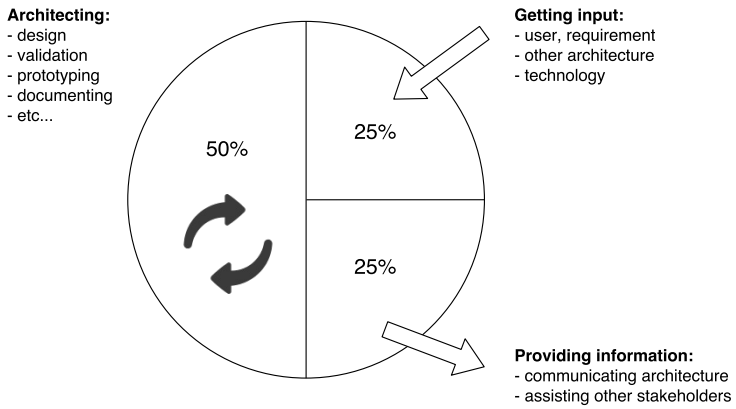


Figure 3.9: Time division of an architect, adapted from [92].

Fowler [50] mentions two types of persons, coming back in the roles of the model of Klein [87] and in Bredemeyers [19] work: *Architectus Reloadus* is the architect as the person who makes the important decisions at the start of a project, so that everyone has a plan to follow and the conceptual integrity of the architecture is maintained. As an *Architectus Oryzus*, the architect is very aware of what is going on within the project and the teams. His focus is intense collaboration with all parties involved.

3.8 Conceptual Model

The discussed concepts in this chapter are best illustrated using a conceptual model. The conceptual model is based on the ISO/IEC/IEEE standard 42010 [74] and articles cited throughout this chapter. The conceptual model starts with a contextual view on a system. A `system` is situated in its `environment`: the environment determines the factors (developmental, operational, technical, political, regulatory, and many other) that exert influence on the system, and the environment could include other systems. Furthermore, a `stakeholder` has interest in a `system`, regardless of what that particular interest(s) may be Sect. 3.2. Those interests are called `concerns`. A `system` exhibits an architecture, no matter whether that is documented or not: the `architectural description` expresses or represents the architecture [74]. As we have discussed, the architecture consists of `architecture elements` and their interelement relations. A `stakeholder` has interest in a `system` but the `system` also addresses the needs

of a stakeholder. An architecture description comprises a number of views and viewpoints (see Sect. 3.5). A view comprises multiple architecture models; models that describe the architecture from a certain point of view. A model conforms to a set of conventions specified in a model kind. A model kind is documented in a viewpoint: the viewpoint comprises of model kinds that are needed for the models that are used in the views. A viewpoint frames one or more concerns, and a view addresses a specific series of concerns that a stakeholder has.

This information, combined with the conceptual model of Tang, Jin and Han [141] (Fig. 3.5 in Section 3.4), leads to a conceptual model presented in Figure 3.10. In this model, a concern is a centric element. As a result, a *concern-driven* conceptual model of software architecture can be developed. A system of interest exhibits an architecture, which is expressed in an architectural description (AD). In the AD, concerns are identified. Each architectural viewpoint frames one or more concerns. Each view addresses one or more concerns. A concern motivates a architectural design decision (ADD), and an ADD pertains to one or more concerns. Requirements manifest concerns. ADDs can depend upon other ADDs, and result in design outcomes. A design outcome is represented as an AD element.

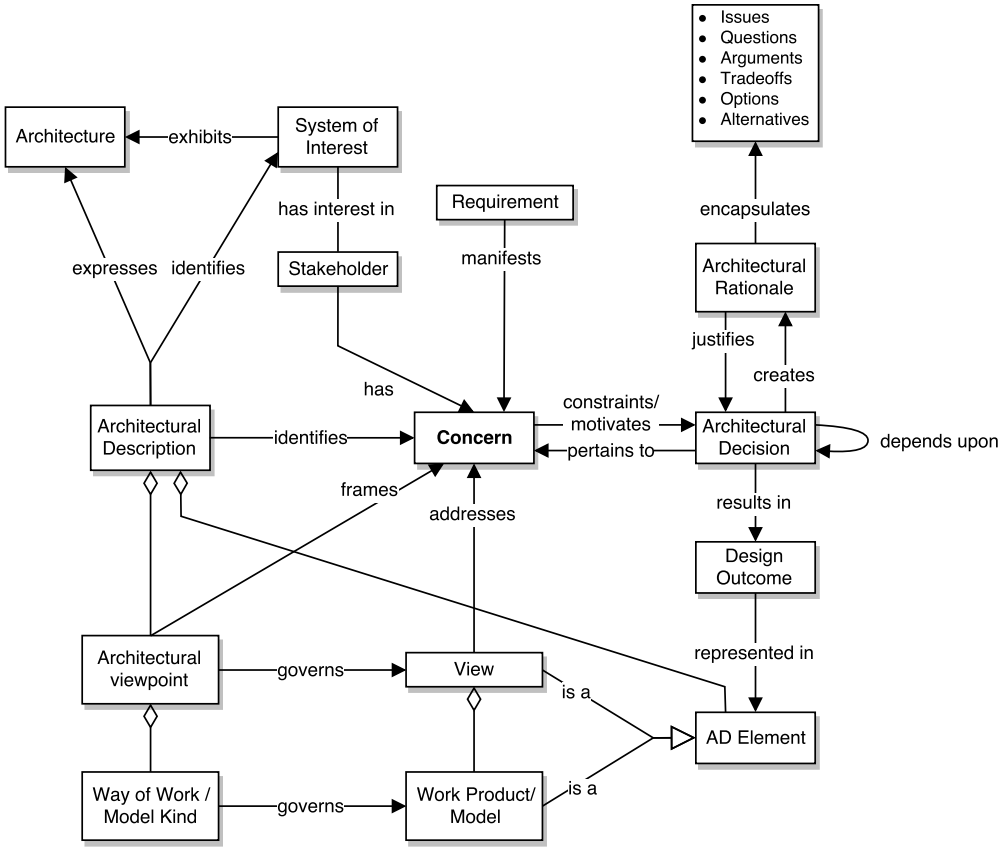


Figure 3.10: A conceptual model of all the notions related to and cross-cutted by concerns.

This concern-driven conceptual model presents a sound, unambiguous foundation for a thorough understanding of how main architectural concepts are related. This will provide guidance during the construction of the CDIM method.

3.9 Summary

The term software architecture is used for reasoning about software systems, presenting a helpful abstraction for managing complexity in such as system. Software architecture refers to the set of critical and early design decisions about the system. Stakeholders are the most vital component of software architecting as they affect the architectural decision decisions, thereby driving the shape and direction of the architecture. Their interests and concerns for a system are referred to as the (stakeholders') concerns of a system. There are thousands, and an architect should understand the nature, source and priority of the concerns as early as possible. We have defined three types of concerns, architectural goals, functional architectural requirements, and quality requirements. Concerns form the primary drivers and building blocks of a software architecture, and force architects to think about important aspects when devising an architecture. As a results of various concerns, architects have to make architectural design decisions, which lead to design solutions. There is a spectrum of design decisions: high-level design decisions require informal notations while low-level decisions can benefit from formal notations. As principal and early design decisions are usually high-level, informal models are common in the domain of software architecture. The process of architecting - or architecture definition - is a life cycle practice that connects problem space and solution space. It consists of architectural analysis, architectural synthesis and architectural evaluation, in order to translate problems and concerns to solutions and designs in the form of an architecture. The role of the architect is ambiguous. It is context- and project-dependent, changes over time, but minimally stretches the definition of the architecture, maintenance of the conceptual integrity, deciding on ASRs and ADDs, assessment and mitigation of risks, participating in planning and development, consulting, informing and collaborating with other stakeholders, and participating in development of a system's vision.

The aim of this chapter was to discuss and present knowledge gained through extensive literature research. The chapter has discussed several concepts that will be used during (1) research on inconsistency management in software architecture, (2) development of the questions for the expert interviews, (3) the Method Association Approach, and (4) development of the CDIM method.

4

Background: Inconsistency Management

This chapter aims to identify inconsistency and inconsistency management in software architecture, and addresses several means and techniques to detect inconsistency. It builds on the concepts presented in the previous chapter, and presents an answer to the second and third sub research questions.

4.1 Inconsistency

Many authors present definitions for 'consistency' and 'inconsistency' [16, 18, 139]. We do not adopt one of their definitions, but extract the important concepts from the definitions and present them. Boucké defines consistency as a property of the relation between views and refers to consistency as the state in which information in two or more views is mutually compatible, e.g. does not conflict [16]. Bowman states that "*two specifications are consistent if and only if it is possible for at least one example of a product (or implementation) to exist that can conform to both of the specifications*" [18, p. 10]. Consistency is defined by Spanoudakis and Zisman as "*a state in which two or more elements, which overlap in different models of the same system, have a satisfactory joint description*" [139, p. 3].

'Inconsistency' can thus be seen as present if two or more statements are made that are not jointly satisfiable [64], are mutually incompatible [34], or do conflict [16]. There are various examples of inconsistencies: failure of a syntactic equivalence test, non-conformance to a standard or constraint [64], two specified requirements that assert something that is impossible, or two developers implementing a non-relational and a relational database technology for the same database.

4.2 Dimensions of Inconsistency

The examples show that inconsistency has different dimensions and manifests itself in many various forms. This section addresses an extensive discussion of different forms of inconsistency relevant in the domain of software architecture. Inconsistency in software architecture has a wide range of dimensions. Dependent on the context and the focus of the architect, one is more relevant

than the other. Example forms of inconsistency may be inconsistency in code [80], inconsistent requirements or concerns [143], and model inconsistency [13,135]. We view these as the '**tangible**' forms of inconsistency.

As Figure 3.5 in Sect. 3.4 shows, *concerns* (motivational reasons) cause design decisions, in their turn resulting in design outcomes. So when concerns or design decisions are conflicting, we can state that they are inconsistent, following a definition of Dashofy [34, p. 58]: "*design decisions are consistent if they make mutually compatible (i.e., non-contradictory) assertions*". We see that these '**intangible**' inconsistencies are often denominated as *conflict*, when they are still undocumented or unspecified. That is, an intangible inconsistency 'conflict' [143] has the potential to manifest itself as a 'tangible' inconsistency, due to the fact that design solutions follow from design decisions, in their turn caused by design concerns [142,143]. Intangible conflicts can manifest themselves in the design or even the implementation as tangible inconsistency, resulting from those design decisions.

This corresponds with the view of architectural design decisions as first-class entities [91,95] (see Sect. 3.1). For many years, architecture research has focused on the consequences of design decisions, often captured as the 'design' or 'architecture' of the system, contained in views [24,74,129] or architecture description languages. But in doing so, knowledge about the decision and the rationale behind the decision is lost [91]. If we adopt the definition that a software system's architecture is the set of principal design decisions about the system [34,105], we see that inconsistency - even though it may be undocumented (intangible) at this point - already may emerge if design decisions are inconsistent or contradictory. Tangible inconsistency then manifests itself in model inconsistency or code inconsistency for example. This is supported by Spanoudakis [139] who defines the cause of inconsistency 'as the conflict(s) in the perspectives and/or goals of the stakeholders'.

As discussed in Section 3.6, the architecture process connects the problem space and the solution space. Inconsistency appears in elements in the problem space, during the architecting process, and in elements in the solution space. Table 4.1 presents a comprehensive but non-exhaustive overview of which forms of inconsistency arise in the problem space, the architecture process, and the solution space (indicated in the table headers and similar to the ovals in Fig. 5.1, in Chapter 5). The following sections discuss in detail the important concepts of inconsistency (management).

4.3 Forms of Inconsistency

Types of inconsistency are discussed by many different authors in entirely different ways. Thoroughly classifying the types of inconsistency would be a topic of study on its own. As a result, we present several axes on which inconsistency can arise.

4.3.1 Inter-model and intra-model inconsistency

One of the properties of a single model is intra-model inconsistency. If a model's elements are not syntactically or semantically correct, or in any way contain conflicting assertions, the model contains intra-model inconsistency. On the other hand, inter-model inconsistency is a conflict between different (related) models. These models can usually be related to each other by a transformation relationship [44]. A transformation refers to the actions needed to create a model

	Problem Space	Architecture Process		Solution Space
Inconsistency appears in:	Requirements, models, user stories.	Concerns, design problems, design decisions.	Models, views, design solutions, notations, diagrams.	Code, implementation, packages, libraries.
Forms of inconsistency:	Contradicting requirements, use case inconsistency.	Conflicting design decisions or solutions [34].	Horizontal / vertical [72], syntactic / semantic, inter- / intra-model [44].	Type, version, derivation, link, reachable inconsistency [80].
Inconsistency detection mechanisms:	Prioritized merger [106], viewpoints (structured requirements templates) [48].	Software architecture review by association [143], human centered approaches [139]	Formal specification languages [138], model checking algorithms (e.g. SPIN [71]), human-centered approaches [41, 127].	Tests, code reviews, parsers, compilers, configuration management [96, 113].
Difficulty of detection of inconsistency:	Easier when formally specified, harder when informally specified.	Difficult due to different abstraction levels, granularity and formality [34].	Easier when formally and homogeneous specified, hard when informally and heterogeneous [139].	Easier using compilers, tests, parsers, lexical analyzers [80, 96].

Table 4.1: Comprehensive but non-exhaustive overview of inconsistency appearances, forms and detection mechanisms.

from another model. Usually, inter-model inconsistency arises when a model of a lower abstraction type is constructed from a more abstract model.

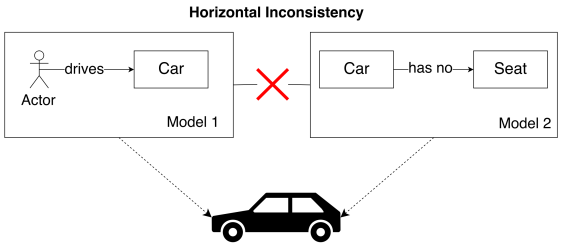


Figure 4.1: Horizontal inconsistency.

4.3.2 Horizontal and vertical inconsistency

Vertical inconsistency is inconsistency between two models that have different abstraction levels, such as the refinements of a data model [72]. Produced models should be consistent with models at more abstract levels [72]. If there is no definition of refinement relationships, than vertical inconsistencies can arise, because models could be refined in numerous ways. As this dimension of inconsistency concerns two or more models, the term *inter-model consistency* is also used. Vertical inconsistency is usually overlooked in approaches to handle inconsistency problems [99]. The very simple example in Fig. 4.2 describes two data models of a car. Model 1 and Model 1.1 are vertically inconsistent, because where Model 1 declares that a 'car' should have a 'wheel', Model

1.1 describes that a 'car' has 'wings' instead. The refined Model 1.1 is not mutually compatible with Model 1.

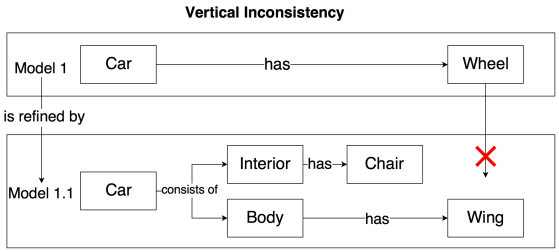


Figure 4.2: Vertical inconsistency.

On the contrary, horizontal inconsistency emerges in models of the same abstraction level. Huzar [72] states that horizontal inconsistency is caused by the fact that some models do not have precise semantics, resulting in possible ambiguous implementations of a model. For example, horizontal inconsistency is expected between elements of a model representing a static and dynamic view [72], on the same abstraction level. These specifications should be consistent and not contradictory because an implementation from two inconsistent specifications would be infeasible [45]. The simple example in Fig. 4.1 depicts horizontal inconsistency: the use case model (Model 1) depicts that a 'driver' drives a 'car', but Model 2 does not prescribe a 'seat' in the 'car'. Implementing both specifications would result in a product that is infeasible. Both examples are kept utterly simple on purpose.

4.3.3 Inter-phase and intra-phase inconsistency

Inconsistencies in the description of a system may exist between elements of different phases (*inter-phase*), as well as between views in a *single* phase (*intra-phase*) [108]. An example of the prior is consistency between the models used during design and the code used in implementation (inter-phase consistency), for example between a prevailing class diagram and a subordinate C++ code fragment. An example of the latter is consistency between a class diagram and a message sequence diagram, both used in detailed design of a system.

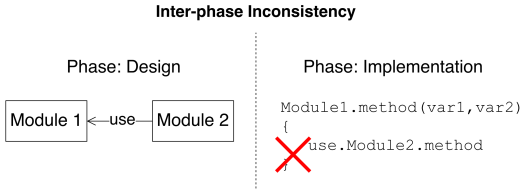


Figure 4.3: Inter-phase inconsistency.

The simple example in Figure 4.3 clarifies inter-phase inconsistency. Taken a small functional design and implementation in the form of some source code, it can easily be seen that the source code is not consistent with the design. The design states that Module 2 should use Module 1, while the source code does not adhere to the design. Of course, it is acknowledged that there are

various other forms of inconsistencies between phases, but the example is purely to illustrate the concept.

Graphically, intra-phase inconsistency can be illustrated the same way as in Figure 4.1, if the two models are the result of the same phase. Intra-phase consistency is related to the vertical and horizontal consistency dimensions, as two models can be both horizontally intra-phase consistent (a) and vertically intra-phase consistent (b). This has been illustrated in the example in Figure 4.4. The characters $r_1, C_2, C_{2.1}, MS_1, MS_{1.1}$ and the fragment source code `Module1.method` are partial specifications of the system under analysis and represent the requirements, class diagrams, message sequence diagrams and source code respectively.

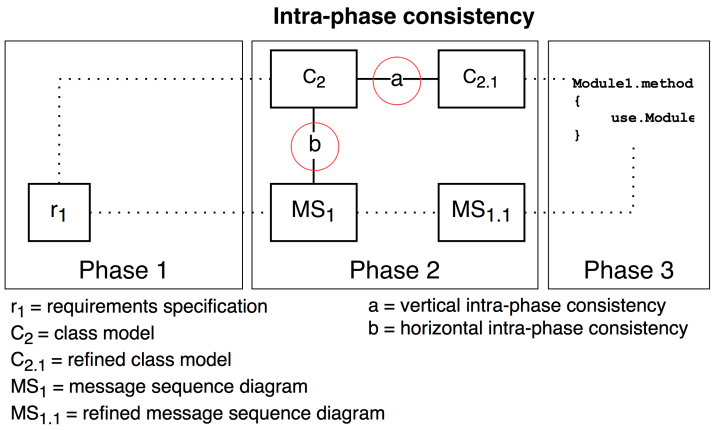


Figure 4.4: Intra-phase inconsistency.

4.3.4 Architectural drift: inter-phase inconsistency

No matter how much effort is spent on the design of the architecture of a system, it is inevitable that the resulting source code (*implementation*) is going to diverge from the intended architecture over time [107]. This divergence of the source code and the initial high level design of a system, resulting in certain discrepancies, is called architectural drift [128]. If the implementation diverges away from the design, all the effort spent on designing the system may be lost [134]. It is acknowledged that architectural drift may contribute to increased costs and difficulties in system maintenance [63, 107]. Moreover, architectural drift could result in the situation where the architecture description is of "less use or possibly even harmful for developers" [128, p.1], referring to [63, 107]. It can confuse maintainers if they have to work with two inconsistent versions of the system (the realized architecture in the form of source code and the intended architecture in the form of documentation), which may lead to increased maintenance costs and time [43]. Work of Singer [137] has shown that developers are likely to rely more on the source code because they do not trust design documents. Bass et al. [8] state that architectural drift can be prevented by having strong management and process discipline. They present three alternatives for achieving code alignment with the architecture: a) *sync at life-cycle milestone*, at the point of a release, the architecture is updated to changes in the source code; b) *sync at crisis*, the architecture is rewritten at the point of an technical impasse; c) *sync at check-in*, which means that the

architecture is strongly codified and used to vet any check-in.

4.3.5 Syntactic and semantic inconsistency

Syntactical inconsistency is the state in which a specification does not conform to the syntax that is provided by the meta-model (or the model kind) [45]. For instance, a model specifying a generalization relation between two model elements while the meta-model only allows association relations is not syntactically consistent. It thus refers to the well-formedness of a (single) diagram or view [60]. Semanctic inconsistency can be best explained by discussing semantic *consistency*. Semantic consistency is a stricter type of consistency which requires syntactic consistency to already be present. It requires different but related models to be semantically compatible with respect to the aspects of the system that the models describe [45]. With regards to semantic inconsistency, also horizontal and vertical inconsistency exist: vertical semantical inconsistency is a state in which a refined model is not mutually corresponding to the more abstract model it refines [45], whereas horizontal semantical inconsistency is the state in which two related views, with the same abstraction level, have mutually incompatible specifications with regards to the systems' aspects they describe. Semantic *consistency* relies on the underlying semantics of the models that are under analysis. Semantics refers to the presence of a mathematical model that defines the meaning of a diagram, while allowing one to reason and infer on the specification of a diagram [60].

4.4 Inconsistency Causes

Inconsistency arises inevitably due to the fact that software architecture is concerned with heterogeneous, multi-actor, multi-view and multi-model activities [47,59,113,116]. Finkelstein [47] argues that a major cause of inconsistency is the fact that multiple actors need to work together, each with different views, opinions and interpretations. For instance, different stakeholders involved in architecture use different languages or development strategies, address different stages of the development, have partly overlapping or non-overlapping areas of concern, and have different technical or economic objectives [113]. In addition, systems are described using multiple views, which are thus related [16, 108]: these relations are another source of inevitable inconsistency in the architecture description [8, 99, 129].

Due to this heterogeneity and the diverse context of software architecture, inconsistency management is inherently difficult [59, 102, 115]. In addition, a lot of architectural knowledge is contained in the heads of involved architects and developers [93] and documentation of architectural design decisions (ADDs) [159] and the architecture design is often not maintained over time [77, 128]. Architectural information that is available, can be specified in a wide range of formats such as ad-hoc informal box-and-line drawings, more formal models, or semi-structured text documents. Though inconsistency management is needed, the possibilities for managing inconsistency in software architecture are limited [108], and architects thus benefit from methods and infrastructures that aid them in management of inconsistency.

4.5 Implications of Inconsistency

Inconsistency is prevalent and ubiquitous in software development and software architecture (SA) [56]. Inconsistency in software architecture is not necessarily a bad thing [111, 139]: "*Inconsistencies highlight conflicts between the views, perceptions, and goals of the stakeholders involved in the development process (which must be dealt with in an accountable way or, otherwise, they may put in risk the acceptance and usability of the system), indicate aspects of the system which deserve further analysis, and facilitate the exploration of alternatives in system development and the elicitation of information about the system.*" [139, p. 2].

However, some inconsistencies that remain undiscovered can lead to all kinds of problems [72, 108, 116]. For example, inconsistencies may cause delay and therefore increase the costs involved in developing the system, compromise the system's quality attributes (reliability or safety), or make it difficult to make changes and maintain the system (e.g. when design documents are inconsistent, certain design decisions may be unclear for architects not working on the system from the beginning). Another (historic) example is the Ariane-5 crash (flight 501) [114]. Ariane 5 reused parts of the software from Ariane 4. In Ariane 4, an inconsistency had been detected between the requirement that all exceptions must be handled, and the implementation in which some exception handling was switched off for various exceptions in order to meet memory and performance concerns. In Ariane 4, the inconsistency was tolerated, as the analysis concluded that the overflow would never occur, and so the exception handling could stay switched off, and thus the inconsistency was tolerated. However, the Ariane 5 had a greater horizontal acceleration than the Ariane 4 types. It caused the rocket's system to crash and transmit data that was misinterpreted by the autopilot as fake velocity data. The larger horizontal acceleration resulted in a data conversion from a 64-bit floating point number to 16-bit integer value, which caused an overflow-error and caused a hardware exception. These exception handles were omitted in the Ariane 4 design. The error halted the navigation system, eventually resulting in the destruction of the Ariane 5 flight.

4.6 Towards Inconsistency Management

The positive implications of inconsistency together with insights from practice demonstrate that maintaining consistency at all times is counter-productive [116] and in some cases inconsistency is acceptable or even desirable [14, 47]. In many cases inconsistencies reflect minor errors or carelessness which could relatively easy be resolved. Other inconsistencies are the result of serious conflicts of interests with substantial consequences. In those cases, inconsistencies can not, or should not, be automatically corrected [59]. So, rather than maintaining consistency at all times, the focus should be on 'managing' inconsistency: identifying inconsistency, preserving inconsistency where it is acceptable, and deferring or solving inconsistency when it is needed [47, 115, 116].

4.7 Inconsistency Management

Spanoudakis and Zisman [139] propose a framework for inconsistency management (IM), based on [49] and [115]. The framework consists of 6 activities: *overlap detection*, *detection of inconsistencies*, *diagnosis of inconsistencies*, *handling of inconsistencies*, *tracking of findings*, and

specification of an inconsistency management policy. The particular focus of the framework is on model inconsistency, as model inconsistency is the most common form of inconsistency in literature. The following sections briefly discuss the first five of the six steps, as the last step lies outside the scope of our research.

Overlap detection. Detection of overlaps is a critical component of IM, since *overlap* is a precondition for inconsistency [49, 138]. Overlap emerges due to different views, assumptions, and concerns all being interrelated because they are related to the same system [108]. The presence of such interrelations introduces the potential for inconsistency [139]. Overlapping concerns can result in mutually incompatible design decisions [142,143], architecturally significant requirements, or design solutions, which will be eventually translated in (possibly inconsistent) models, diagrams or other specifications. There are many related approaches for detection of overlaps. Techniques that focus on *detection of overlaps* do this based on for example representation conventions, shared ontologies, or similarity analysis [139].

Representation convention forms the basis for several unification algorithms [88] in first-order logical languages [139]. One of the most common representation conventions is to assume an overlap between model elements when they have identical names and no overlap between any pair of elements when this is not the case [139]. An alternative approach to identification of overlap is the use of shared ontologies. These approaches require a model's authors to tag the elements in them in a shared ontology. However this requires the definition of each item included in the ontology, and ontologies can be inconsistent themselves [139]. Similarity analysis consists of relating models through a common meta-model. This approach employs the constructs that imply existence of overlap relations, often included in modeling languages [139]. In strict entity-relationship models for example, the 'is-a' relation is a statement of overlap which can be used to detect overlaps. A limitation is that this approach is sensitive to model heterogeneity.

One last technique for overlap detection is to let stakeholders identify and define overlaps between elements. This is a particularly appropriate technique when models are of different abstraction levels, when different modeling styles are used, or when models are heterogeneous [34]. Using correspondence relations, stakeholders can define certain relations between elements: for example, in the Views and Beyond approach (architecture documentation using several views) [24] authors of an architectural documentation (AD) use correspondence rules to indicate overlapping or related views. Correspondence rules are also advocated by the ISO/IEC/IEEE 42010 [74].

Inconsistency detection. Inconsistency detection should be performed after identifying overlaps between elements. Techniques for *detection of inconsistencies* are logic-based approaches, model checking approaches, specialized model analyses, and human-centered collaborative approaches [139]. Logic-based approaches are characterized by the use of formal interference techniques to derive inconsistencies from models, that need to be expressed in formal modeling languages (e.g. first-order classical logic [138]). The advantage is that formal theorem proving has is that the reasoning rules are well defined and have sound semantics. The limitation is that these approaches are computationally inefficient and require formalization of models. Model checking approaches use algorithms to check the compliance with consistency rules (such as SPIN [71]). For model checking approaches, it must be possible to translate models in a state-oriented language used by the model checker, and only a limited amount of consistency rules can be checked. Other approaches are specialized ways of checking inconsistency, most of these check the satisfaction of specific consistency rules ([111]). Checking for inconsistency requires models to be expressed in a specific common language (e.g. graphs, XML or Petri nets). These approaches have their

limitations in the sense that they enable only specific kinds of consistency rules to be checked [139], and require the architect to build (large) rule sets [116]. All of the approaches suffer from the heterogeneity in models and actors of the SA environment, or involve extensive formalization or documentation, which is time-consuming. In addition, most of the approaches concentrate on one particular type of description [115]. Rule-based approaches Formal approaches permit a wide range of applications due to their mathematical foundations, but are not yet very popular in the industrial practices [10]. Main reasons of the unpopularity of these approaches are that they are difficult for the modelers to use, and the feedback is usually poor and difficult to grasp for industrial users [99]. In addition, they are usually poorly suitable for informal models.

Inconsistency diagnosis. Inconsistency diagnosis is concerned with the identification of the source, cause and the impact of an inconsistency [139]. The source of an inconsistency refers to the elements in which a conflict is detected, or the elements that violate a consistency rule [115]. As stated earlier (see Sect. 4.2), the cause of an inconsistency in the framework of Spanoudakis is the conflict in the perspectives or goals of the involved stakeholders. Finally, the impact of an inconsistency is defined as the effects an inconsistency has on the system in question. Diagnosis of detected inconsistency is an important step in the IM process as the source, cause and impact of an inconsistency affect the associated priority of handling an inconsistency. Furthermore, the diagnosis determines the options that are suitable for handling an inconsistency, and influence the cost and benefits of solving (or not solving) an inconsistency.

Inconsistency handling. Handling inconsistency is concerned with the identification and execution of the possible actions for dealing with an inconsistency. The handling of inconsistencies is seen as a central activity in inconsistency management. This activity refers to: (a) identifying available actions for dealing with inconsistency and (b) selecting one of those actions to execute.

Tracking. Tracking refers to recording important information of the inconsistency in a certain knowledge base [139]. This activity is concerned with the recording of important knowledge and information gained during the process, such as establishing a rationale for why certain decisions are taken and actions are performed. This activity can add a great amount of overhead to the IM process, so it's contents should be carefully evaluated in relation to the benefits that one expects of the step [139].

4.8 Early Detection of Inconsistency

The earlier inconsistency is identified, the cheaper it is to fix [64]. At early stages in the architecting process, architects deal with coarse-grained models and high-level design decisions, which are usually recorded and documented using more informal notations [34]. Considering the design decisions that address the system's aspects in the same terms and at the same level of detail, it is less complicated to identify and detect inconsistency (as design decisions are comparable in the sense that they make assertions such as 'X and not X'). Considering design decisions that express different levels of precision or abstraction, identifying inconsistency is not so straightforward [34]. This preferably involves stakeholders and their judgments. As a result, formal and model-checking approaches (discussed in Sect. 4.7) for IM are less applicable in the context of SA. That emphasizes the need for a method that detects conflicting concerns and high level design decisions, as these give occasion for 'early' inconsistencies in a SA, which are less costly and also may be less consequential.

4.9 Human-Centered Inconsistency Detection

Approaches that are more applicable in this context are architectural reviews and human-centered collaborative exploration (such as Synoptic [41] or DealScribe [127]). These techniques assume the detection of inconsistency is the result of a collaborative inspection of several models by stakeholders [139]. A limitation of these types of approaches is that they are time-consuming and labor-intensive, and become difficult in the context of large models and systems. In Synoptic, stakeholders fill in so-called 'conflicts forms' to describe a conflict that exists between overlapping elements in a model. In DealScribe, the stakeholders identify conflicts between so-called 'root requirements' in their models. Root requirements are identified for concepts present in the models, and pairwise analysis of possible interactions between root requirements results in a list of conflicting requirements.

4.10 Inconsistency Management as a Verification and Validation Discipline

Inconsistency management is a part of the process of verification and validation [64]. Verification and validation of an architecture is carried out with the use of architectural evaluation (AE) methods [4]. The primary objective of AE methods is to assess an architecture's potential to deliver an adequate system, thereby fulfilling required quality attributes and to identify risks [97]. This indicates a certain similarity with the purpose of inconsistency management. Scenario-based approaches, which are a sub-category of AE methods, are developed over the past decade and have proved to be mature [5]. In this sense, they form a source of inspiration for the CDIM method developed in this thesis. Fragments of the following methods have been included in the development of the CDIM method: SAAM [84], ATAM [85], APTIA [83], ALMA [12], SAAMER [101], SBAR [11], and TARA [155] (see Babar [5] and Dobrica [39] for a comparison of various AE methods). The AE methods that form a basis for the CDIM method are discussed in Section 6.5.

4.11 Summary

Inconsistency is present if two or more assertions are made that are conflicting, not jointly satisfiable, or mutually incompatible. Inconsistency is prevalent in software architecture, and arises due to the fact that software architecture is concerned with multiple models, multiple phases, and work is often coordinated among multiple actors; each with their own perspectives, views and opinions. Although inconsistency is not necessarily a bad thing, undiscovered inconsistencies can lead to all kinds of problems. It is acknowledged that inconsistency discovered early in the process is easier and more economic to fix or adapt than later in the process. There are many dimensions to inconsistency, of which we distinguish between two important ones: tangible and intangible inconsistency. Tangible inconsistency refers to inconsistency in models, specifications, code, and documentation. Intangible inconsistency refers to inconsistent assumptions, concerns, or design decisions. Inconsistency arises in various forms during the architecting process: in the problem space, inconsistency arises in for instance requirements, models and user stories; during

the architecting process, inconsistency in design decisions, assumptions and models is important; in the solution space, inconsistency in code and models deserves the focus. Architects have to deal with inconsistency in informal models, as these are often used for capturing high-level design decisions. Among tangible forms of inconsistency, various types exist: inter- and intra-model inconsistency, horizontal and vertical inconsistency and inter-phase and intra-phase inconsistency. Inconsistency management practically consists of detecting overlaps, detecting inconsistency, diagnosing inconsistency, and handling inconsistency. Overlap detection is critical, as overlap is a precondition for inconsistency. Means and techniques for detecting inconsistency in tangible forms of inconsistency are numerous. However, for informal models or intangible forms of inconsistency, human-based collaborative approaches are needed.

5 Requirements of the CDIM Method

This chapter consists of two parts: first, it discusses six expert interviews and their results. These results are in the shape of 39 advices and 11 requirements. Second, it presents the set of final requirements, that consists of 23 requirements from the interviews and from literature. Together, the advices and requirements form the input for the Method Association Approach (MAA) and the development of the CDIM method.

5.1 Method

To gain more insight in inconsistency management in practice, six practitioners have been interviewed. The interviewees were selected based on their job title (software architect), experience (5+ years), and the type of business (software producing organization or business software). Details are included in Table 5.1. Organization size is measured in amount of employees, and experience refers to experience as software architect, or leading role in development. Objectives of the interviews were: (1) to learn about inconsistency management in practice, (2) to discover areas of improvement, and (3) to identify and evaluate feature groups (input for MAA), and (4) to identify requirements for the CDIM.

#	Org. size	Org. type	Job title	Years of experience
1	Large (2.000+)	IT Services	Principal IT architect	20 years
2	Large (2.000+)	IT Services	Software architect	25 years
3	Medium (350+)	ERP Software	Lead software architect	13 years
4	Large (700+)	Online retail	Lead IT architect	07 years
5	Small (50+)	Healthcare IT	Software architect	10 years
6	Large (2.000+)	Business software	Principal software architect	30 years

Table 5.1: The background information on participating experts.

The data was collected in semi-structured qualitative interviews. Semi-structured interviews enable the researcher to expand or extend on relevant topics, resulting in possibly more detailed insights. Open ended questions have been phrased to structure the conversation, and to assure

that participants could talk in-depth about their experiences. The coding behind the interviews was inductive, which means that coding is based on the results of the interviews. To ensure valuable data, the participants were informed about the topic and the aim of the study, their role in the study, and how the results would be applied. The interviews were recorded with permission, and transcribed as soon as possible after each interview, by all means on the same day as the interview took place. The transcriptions of the interviews were subsequently sent to the respective participant for verification.

Subsequently, each transcription was carefully analyzed and valuable knowledge was transferred to a new document. In total, 158 quotes and advices of the participants were extracted from the transcriptions. They have been categorized into the following classes:

- *Inconsistency Management*
  - *Inconsistency Causes*
  - *Scoping and Planning*
  - *Inconsistency Monitoring*
  - *Inconsistency Classification*
  - *Inconsistency Handling*
- *Architecture Rationale*
- *Situational Factors*

Further advices were in the form of *method requirements* which will be discussed in Sect. 5.3.

5.2 Results: Advices

Within each category, advices have been sorted, bundled and carefully rewritten. This resulted in a total of 50 advices: 7 advices on Situational Factors, 30 advices on Inconsistency Management (in total), 2 advices on Architecture Rationale, and 11 *requirements*. The 32 advices on Inconsistency Management and Architecture Rationale help to understand the process and difficulties of inconsistency management. The 7 advices on Situational Factors will improve the ability to turn the method to a situation at hand. The advices that were in the form of *method requirements* are used for the development of the method, and described in Section 5.3.

Each advice is 'tagged' and numbered with an abbreviation that matches their category. IMA1 equals the first advice on Inconsistency Management, and so forth. Every statement or quote produced by an expert is referenced by adding (Exp.X), or (Exp.X,Y) if multiple experts mentioned a particular issue. The numbers in the round brackets refer to the number of experts that brought up a certain advice: for example, IMA3 was mentioned twice (2).

5.2.1 Inconsistency Management

Experts provided insight on important factors that relate to 'inconsistency management':

- **IMA1: Concerns change and evolve during the project.** It is wise to often reconsider concerns. "Using them as first-class entities is a good idea" (Exp.2).

- **IMA2: Do not display concerns using numbers (3).** If you wrap up concerns using numbers, people have to look up the numbers in a separate list, which will get forgotten quickly (Exp.2,3,4).
- **IMA3: Automate as much as possible (2).** When managing inconsistencies, it is important to look for activities that could be done by tools or with the help of tooling.
- **IMA4: Inconsistent specifications could be destructive.** "Especially when requirements change or are 'quick fixed', tests and testers could test the outdated specifications, while there are new ones. This could lead to undesired costs." (Exp.3)
- **IMA5: Carefully decide how much documentation is needed.** Every company is unique in the need for documentation. "Documentation is worthless if it is not read, or updated. If something is documented very carefully, but 1) nobody reads it, 2) nobody updates it, than documentation is worthless" (Exp.2). "In general, documentation is running behind. The code is our skeleton." (Exp.3)

Inconsistency Causes

An inconsistency is caused by various factors. Experts mentioned a collection of inconsistency causes:

- **ICA1: Document important decisions.** "Inconsistencies especially arise when preceding decisions have not been documented, especially in the case of pragmatic architecture 'on the fly'." (Exp.5)
- **ICA2: Prevent unaligned assumptions about the same element (2).** "Inconsistencies arise between overlapping elements. If you take a decision on the basis of an assumption, while there is a conflicting assumption that is irreversible, you have yourself a problem." (Exp.4x)
- **ICA3: Inconsistencies arise when dealing with multiple stakeholders (3).** It is important to recognize that inconsistencies and conflicts could arise because you deal with varying stakeholders and their concerns.
- **ICA4: Inconsistencies arise between products of different teams.** In an environment with multiple teams, every team is working following their own conventions. "There are no tools to check inconsistencies between the work of teams." (Exp.5). "You cannot exactly know what independent teams are doing. Communication between teams is difficult. Architects should safeguard this." (Exp.6)

Scoping and planning

An important concept within scoping and planning is having involved stakeholders:

- **SCP1: Involve and invite stakeholders (2).** "You have to bring together the stakeholders that have raised the relevant concerns. Very important!" (Exp.1,2).
- **SCP2: Associate identified concerns with stakeholders (3).** It is important to associate the concerns with the stakeholders that have introduced them.

Inconsistency Monitoring

Experts were asked to give advice on the topic of monitoring inconsistency. Their advices are presented below:

- **IMO1: Inconsistencies arise at trade-off areas (3).** "I think they are often to be found at these areas. Where one considers two options, one must make a choice, and someone else will make another consideration and possibly another choice." (Exp.4)
- **IMO2: Detect inconsistencies as early as possible (2).** Improvement and repair will be more costly as time passes. "The later an inconsistency is detected, the more worse it gets!" (Exp.2). "The earlier you discover them, the less impact they have and the less people are impacted by it." (Exp.3)
- **IMO3: "Search for consequences of inconsistencies" (Exp.3) (2).** "Inconsistencies appear at other places than they arise." (Exp.4)
- **IMO4: On code level, you can use checks and tests where possible.** "Tests, checks and code reviews increase consistency, quality and integrity of the code." (Exp.5). "[Moreover,] it creates trust among developers." (Exp.3)
- **IMO5: Use 'rulesets': testcases that transcend multiple components.** "An example is checking whether each component offers a service the exact same way." (Exp.5)
- **IMO6: Discuss existing agreements and decisions for each relevant combination of concerns.** Ask what the agreements are that the team has made so far, and ask which decisions have been made about this area. (Exp.3, Exp.6)

Inconsistency Classification & Prioritization

Inconsistencies that have been discovered need to be analyzed in order to classify them. Advices concerning the classification of inconsistency resulted in the following factors:

- **ICL1: Business rationale (4).** Business rationale is the factor to decide whether to solve an inconsistency or not. Two main elements of a business rationale are often risk and financials.
- **ICL2: Carefully balance the the costs of fixing an inconsistency and the consequences of ignoring one (2).** An example: the business finds a solution for an inconsistency too expensive, and accepts 10% of the customers does not have a specific feature (Exp.4).
- **ICL3: Clearly specify who is administrating inconsistencies.** "If too many people are involved, you tend to lose grip." (Exp.2). Ideally, one role should focus on administering inconsistencies.
- **ICL4: Focus on both the cause and consequence when classifying an inconsistency.** "Ask yourself: what is wrong? What is the cause? Even more important: what is the implication of the inconsistency?" (Exp.4) "If nothing happens, we are not going to solve something." (Exp.1)
- **ICL5: Required effort is an important classification factor.** "Classify inconsistencies on the basis of how much work is required to improve or repair it." (Exp.1) This also means that alternatives for the inconsistency have to be considered.
- **ICL6: Continuity is an important classification factor.** "Try to classify inconsistencies on when they need a fix. If it blocks the process, the way of working, or further

progress to some extent, it needs to be solved." (Exp.5)

- **ICL7: "Do not prioritize following a 'MoscoW methodology'" (Exp.1).** This just does not work: everything will be must-have. Play poker card games. This could also be done with stakeholders.
- **ICL8: Set a project goal.** Every concern needs to be prioritized with respect to that particular project goal. This helps unifying opposing opinions and concerns (Exp.1).
- **ICL9: Focus on decisions that are hard to reverse.** The focus of the architecture process should be on the decisions which are hard to reverse or have major costs, so if one is prioritizing concerns, the focus should be on these aspects (Exp.1).

Inconsistency Handling

Experts provided various insights in how to handle inconsistency:

- **ICH1: Create a clear conceptual image of a detected inconsistency.** If critical inconsistencies have been found, ensure everyone involved has a clear conceptual image of the problem (Exp.5).
- **ICH2: Discuss major inconsistencies with other architects and stakeholders (2).** "In the end, the entire team is responsible for a good product, making it crucial to discuss important problems." (Exp.2)
- **ICH3: "Use root cause analysis for critical inconsistencies" (Exp.5).** As the cause of an inconsistency is often not clear, root cause analysis could help finding the origin of a problem.
- **ICH4: Organize knowledge sessions.** In order to foster communication and improve consistency between teams, knowledge sessions form an added benefit. (Exp.6). "Architecture is about communicating and transferring the lessons learnt." (Exp.5)

5.2.2 Architecture Rationale

In many interviews, the architecture design rationale reappeared.

- **ARA1: Having an architecture rationale is beneficial (2).** "Sometimes, when things break down I say to myself, I wish we had taken the time to handle things in a more structural way" (Exp.4).
- **ARA2: Concerns should be linked to important design decisions (Exp.1).** When you have to explain a stakeholder [developer] why something was decided, you can explain him which concern caused the decision. This also aids in doing root cause analysis.

5.2.3 Situational Factors

Some advices on the context of inconsistency management appeared to be contradictory. It appeared that some factors are dependent on the context of the expert: these are called situational factors, and described below:

- **SF1: Type of software produced.** It is dependent on the type of software, in for example medical systems or embedded software, there is more need for explicit documentation (Exp.4).

- **SF2: Development Approach.** Companies using a waterfall methodology have a higher risk of architectural drift, because they are not able to verify frequently (Exp.6). "It is important which architectural process a company has: prescriptive, central architecture document? Or a more descriptive process" (Exp.5). "It is very important to have an architectural blueprint of some kind. However, you do not have to incessantly write out every item, that will only slow you down" (Exp.1). "New developers or engineers want to get a grasp of how the current architecture is implemented, or how it was intended" (Exp.3). These comments show the variety of software development methodologies.
- **SF3: Experience of the architect (3).** An architect needs all his knowledge and experience to truly understand the system and conflicting concerns and you need both will you be able to identify risks (Exp.2,3,6). "[Furthermore], the more experience the SA has, the less documentation and design is needed" (Exp.5). "Understanding important decisions that are difficult to reverse is the most important aspect of a software architect's job" (Exp.1).
- **SF4: Team structure and company size.** The size of the company and the growth of a company is an important contextual factor. Team structure differs from company to company (Exp.5).
- **SF5: Complexity of the environment (3).** Volatile and variable requirements means you should be careful with upfront activities and design (Exp.6). Complexity of the environment is important (Exp.4,5). "We have a lot of politics, regulation, privacy concerns and international standards" (Exp.5).
- **SF6: Culture (2).** "Culture is extremely important. We use the boyscout rule: leave your code better than you find it" (Exp.3). "Listen, be open for suggestions, discuss. It is easier when you have a common goal. Always look for a common goal" (Exp.4).
- **SF8: Architecture rationale.** It is dependent on the context whether you have a documented architecture rationale or not. "In medical devices, embedded software, consequences and hazards of mistakes are important" (Exp.4).

5.3 Results: Requirements

This section presents the requirements [1] for the CDIM method. A part of the requirements resulted from the expert interviews described in the previous section. Remaining requirements have been distilled from literature on two themes: architecture evaluation methods, and architecture description languages (ADLs). They are presented in Sect. 5.3.2.

5.3.1 Requirements from the interviews

The number in parentheses behind each requirement stands for the number of experts that brought up that requirement. In some cases, experts mentioned impediments to adoption of the CDIM method. Impediments have been transposed into requirements.

- **ER1: The method should be simple and consistent in use (3).** This requirement was explicitly brought up by experts 1, 4 and 5. "*In addition, it should force me to look at certain aspects so that I check every element consistently.*" (Exp.4)

[1] referred to as *objectives* in the step two of the DSRM process of Peffers [120].

- **ER2: The instantiation of the method should be specific. (2)** "*The method itself could be abstract, however the tool should be very specific for us.*" (Exp.4,6)
- **ER3: The method should be integrated with an existing tool. (3)** If it is a separate tool, it tends to become 'just another tool'. "*Developers have a lot of tools - really*" (Exp.1). If it becomes a spreadsheet, saved locally, no one will remember or open it (Exp.1,3,5).
- **ER4: There has to be a significant added benefit for the user.** If you can prevent refactoring or prevent the architect from making incorrect decisions, than adoption will come (Exp.3,5).
- **ER5: The architect should not be forced to check and document everything.** "*By forcing an architect to check or document everything, the process is hindered.*" (Exp.4)
- **ER6: The method should be flexible.** Make it 'alive', everything that is inclined towards bureaucracy is abdicated (Exp.4,3). "*Architecture is very dynamic. Try not to expect and design something static.*" (Exp.2)
- **ER7: Incremental improvement and completion**. Have an initial fill, but let the deliverable be improved and complemented continuously. Use the method as a standard way of working, at predetermined moments in time in the process. However, keep it alive, use it as a starting point for a dialogue, not as a document that needs to be filled out (Exp.4).
- **ER8: Prevent cognitive overload.** "*If you are working on a certain aspect, only the relevant related aspects should be visible. If it is too confusing: no one will read it. If it is too long: no one will read it.*" (Exp.3)
- **ER9: Introduce hierarchy.** "*Use hierarchy wherever possible*" (Exp.3). You can then zoom in to the appropriate level of detail when you need it.
- **ER10: Evolution.** Avoid outdated information: if the method results in large quantities of administration, or the deliverable contains outdated information, it will be of no use (Exp.3).
- **ER11: Manage accessibility.** Everyone should be able to use the method and keep track of changes, or edit and update the deliverable (Exp.3).

While some requirements may seem obvious, not satisfying those requirements could likely diminish the validity and the effectiveness of the CDIM method. Others are less evident, but also harder to adhere to. In order to achieve a thorough basis for the method, we also extract various requirements of the CDIM method from literature.

5.3.2 Requirements from literature

Architecture Evaluation Methods

As stated in Section 4.10, inconsistency management is part of the verification and validation process [64], similar to architectural evaluation (AE) methods. This makes AE methods very suitable to extract initial requirements from.

- **AE1: Guidelines and process support.** SA evaluation methods should provide a set of standards and guidelines on when certain actions need to be taken and in which sequence, and on what the deliverables and inputs of the method are [5,58], just like any other software development method.

- **AE2: Approaches.** A good method should explicitly address which techniques (for evaluation) are included, what level of information is required as an input and in which stage the method should be applied [5].
- **AE3: Stakeholder involvement.** Clements et al. [26] argue that active participation of relevant stakeholders is essential for high-quality output of the method. For a successful evaluation, it is important to identify the key stakeholders along with their objectives [5]
- **AE4: Non-technical issues.** SA evaluation methods are a significant part of a software development process, and are thus greatly influenced by non-technical issues like organizational structure, communication, and managerial concerns. The importance of addressing non-technical issues is becoming more apparent [82].
- **AE5: Validation.** In order to guarantee a high validity, a method developer should validate the method by applying various means of validation in software engineering (SE) domain. "*Developing an evaluation method based on personal experiences, general observations, heuristics of the practitioner, or published best practices and claiming that it will work is not good enough*" [5, p.7].
- **AE6: Reusability.** Like other software development method, architectural evaluation is a knowledge-intensive activity which can become very costly if each evaluation starts from the ground up. Reuse is recognized as an important means of gaining quality, productivity and cost-effectiveness in SE domain [3]. Reusability goes beyond simple code reuse, to include design patterns, design rationale, document templates, test scripts, etc [150]. It is thus important to explicitly incorporate activities to manage and utilize artifacts produced by previous iterations of the method.

Architecture Description Languages

Architecture Description Languages (ADLs) are used to describe and represent software architectures, however the widespread adoption of contemporary ADLs stranded [103, 105, 156]. Several authors have investigated the factors that hindered adoption among software architects. As our method should be adopted by architects in practice, it seems obvious to take several of those factors into account as requirements for the CDIM method.

- **AL1: The method should support the modeling of specific (sets of) concerns [105].** The method should make it possible to address specific concerns at once: the scope of architectural concerns grow, the complexity of the work of an architect is increasing: to be sufficiently manageable, architectural models must be divided into subsets that only address specific concerns at once [105].
- **AL2: The method should support multiple views [103].** A lot of ADLs only provide single view support, while practitioners need multi-view support because they need to model a lot of different structures (information, functional, or concurrency views). Support for multiple views is a necessity in ADLs [103], and this multi-view support is therefore a requirement for the CDIM.
- **AL3: The method should fit in an architect's daily practices.** Woods and Hilliard [156] argue that a lot of authors see a mismatch in how architects work and the features provided by ADLs, and present that as an important factor for adoption failure. Therefore, the CDIM method should fit in an architect's daily practices, without constraining him.

- **AL4: The method should not constrain the architectural freedom [105].** Woods and Hilliard [156] state that a lot of ADLs are restrictive and impose a specific model on the architect, restraining his architectural freedom. The CDIM method should support and promote architecting, instead of restraining the architectural freedom.
- **AL5: The method should provide some form of tool support [154].** The majority of the ADLs do not appear to have a software tool that an architect would be familiar with to the architect. Tool support should be made available as extensions to existing tools, such as Visio [154]. A notation should be easily integrated with existing standards and be linked to technologies that practitioners already use [156].
- **AL6: Support for incremental adoption [154].** Comprehensively adopt a new approach in one step is difficult and requires a lot of effort. The chances of success in adoption are higher if architects can apply it incrementally to existing work.

5.3.3 Architecture Principles

Rozanski and Woods [129] identify a set of principles where successful architecture definition processes adhere to. Despite the fact that the CDIM method will not be an architecture definition method, it will play a significant part in the architecting process. These principles will not form direct requirements for the method, but are kept in mind during the development of the CDIM method. The architecture principles (APs) are as follows [129]:

- AP1: It must encourage the effective communication of architectural decisions, principles, and the solution itself to stakeholders.
- AP2: It must be driven by stakeholder concerns. Stakeholders are the reason a system exist, and their input is critical.
- AP3: It must be structured. Given the dynamic nature of architecting it should be structured as much as possible. It must comprise a series of one or more steps, with a clear definition of the objectives, inputs, and outputs of each step.
- AP4: It must be pragmatic. It must consider issues such as lack of time or money, shortage of specific technical skills, and for example unclear or changing requirements.
- AP5: It must be flexible. If the method is flexible, it can be tailored to particular circumstances. This is sometimes referred to as a toolkit or framework approach, with the idea that you use those elements of the toolkit you need and ignore the rest.
- AP6: It must be technology-neutral. That is, it must not mandate that the solution is based around any specific technology, architectural pattern, or development style, nor should it dictate any particular modeling, diagramming, or documentation style.
- AP7: It must integrate with the chosen software development approach, so that existing approaches do not have to be changed.
- AP8: It must align with good software engineering practices, so that it can easily integrate with existing approaches.

5.4 CDIM Focus Area

The theoretical background provides us with a clear view of how the architecting process is situated in its context. Figure 5.1 presents the various elements in which inconsistency can

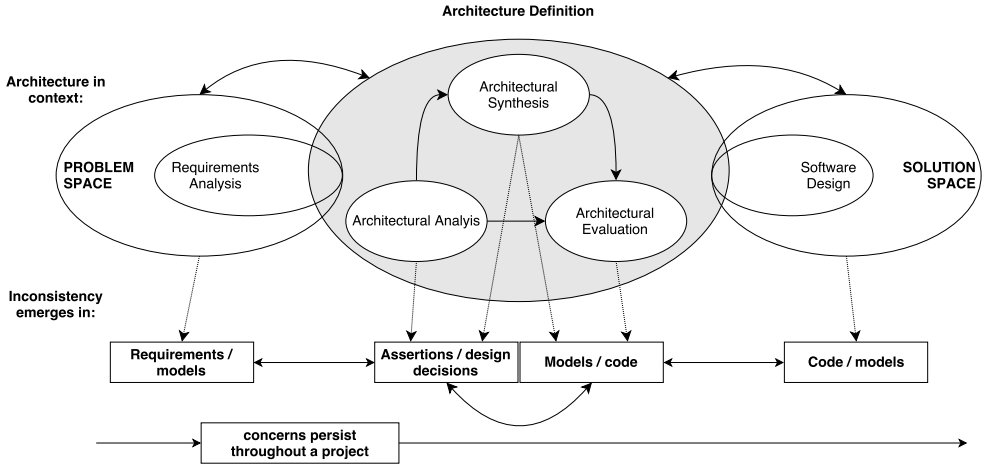


Figure 5.1: Inconsistency arises in different forms throughout the lifecycle.

arise throughout the architecture life cycle. The upper ovals containing 'Architectural Analysis', 'Architectural Synthesis' and 'Architectural Evaluation' represent the architecture definition process [68] as we discussed in Sect. 3.6. The white ovals left and right of the gray oval indicate the problem space and solution space. The problem and solution space are connected by the architecture process (also illustrated in Fig. 3.6). An example of an activity that characterizes the problem space is requirements analysis. For the solution space, software design is an activity that is typically encountered [129]. In Chapter 4 it was demonstrated that inconsistency emerges in different forms, dependent on the phase in the software life cycle. During the requirements analysis, inconsistency arises especially in requirement specifications. During software design, inconsistency arises principally in code and models.

During the process of architecting (architecture definition process) inconsistency is particularly present in design decisions, models and small parts of code. Thus, inconsistency in software architecture refers not only to models or specifications that are inconsistent. Potential interference between different assertions (design decisions) in those products is also referred to as inconsistency [49].

Takeaway: the CDIM method should focus on inconsistency emerging in the architecture definition process.

5.5 Summary

5.5.1 Advices

The six interviews with experts from the field generated interesting insights. Experts had diverse backgrounds, which became apparent in the interviews. Some advices contradicted other advices. For example expert 3 stated: "*in general, documentation is running behind. The code is our skeleton*", while expert 1 argued: "*you need to have an architectural blueprint. This is very important*". The takeaway is that the amount of documentation needed in a project is contextual:

it is important to stress it in the method, but let the architect decide on the intensity and amount of documentation. These kind of advices have been transformed to *situational factors* (Sect. 5.2.3).

Furthermore, none of the participants used a formal or even semi-formal approach for managing inconsistencies. The majority of the experts stated that it would just cost too much time to define rulesets and pursue consistency, using formal methods. As expert 4 put it: "*have short lines of communication. That enables you to detect and solve inconsistencies quickly and ad hoc*".

We learned that the method should not force architects to use a particular prioritization technique, but that it should insist the architect to think about how prioritization must be done. Moreover, the larger part of the architects did not use an explicit architecture design rationale, which is surprising as multiple experts made comments such as: "*sometimes, I wish we took more time to handle things in a more structured way*" (Exp.5). It is important to note that with such a small group of participants no quantitative conclusion can be derived from this observation.

5.5.2 Requirements

The requirements proposed by the experts addressed a wide range of concerns. Simplicity (ER1) was mentioned by three experts, and also addressed in AE literature (AE1, AE2). Tool integration (ER3) was also mentioned by three experts. ADL literature supports this (AL3, AL5), which indicates possible importance. Experts indicated that an architect should not be forces to check and document everything (ER5), backed up by requirement AL4 from literature. The method should be flexible (ER6), which corresponds with requirement AL3. By being flexible, the method can be adapted to particular circumstances and contextual factors. Support for evolution (ER10) is important because outdated information should be prevented. The architecture principles distilled from [129] form appropriate guidelines that help deciding on issues during the development of the method. Literature showed that the CDIM method should have a clear process description, along with guidelines on how to perform those process steps (supported by Architectural Principle 3). Stakeholder involvement (AE3) and reusability (AE6) appeared to be important requirements that need to be satisfied. In addition, the method needs should be validated carefully (AE5). The topic of ADLs revealed that separation of concerns (AL1) and multiple views (AL2) are important issues to account for, because an architecture consist of so many different structures. The method should not limit the architect in his architectural freedom (also supported by Architectural Principle 6) and it should fit in an architect's daily practices (corresponding with Architectural Principle 7).

5.5.3 Overall

The results in this chapter indicate a positive attitude towards a concern-driven inconsistency management method, as concerns were found suitable elements for analysis and discussion, and current inconsistency management practices were ad hoc or non-existent. Reasoning about inconsistency was not performed explicitly and the assessment of risks and evaluation of the architecture in order to find and act upon inconsistencies appeared to be a welcome practice.

A lot of requirements from the expert interviews correspond to the requirements from literature, and clearly indicate the need for a well structured but comprehensive method that aids the architect in detecting inconsistencies. Guided by the architecture definition principles, Chapter 6

describes the development of the CDIM method.

6 Constructing the CDIM method

This chapter describes the efforts of using Situational Method Engineering in combination with the Method Association Approach to construct the CDIM method.

6.1 Introduction

Situational Method Engineering (SME) is the construction and development of a method for a specific development project at hand [22, 62]. SME was introduced in the early nineties [94] with the vision that a method to be used for the development of an information system must be aligned with the context of the project, since each engineering situation is unique and requires a different methodological support [37, 94]. SME arose as a means to construct a specific method for software development. SME provides a well-structured and complete framework for constructing a valid method on a scientific basis and is therefore a suitable approach to design the CDIM method.

According to Ralyté [125] a SME project must have at least two steps: 'set method engineering (ME) goal' and 'construct a method'. Figure 6.1 depicts the overall process with the two 'steps' together with the associated strategies. How each step is achieved depends on the strategy used: for instance, the achievement of 'set method engineering goal' depends on whether the method engineer chooses to construct a method from scratch (from-scratch strategy), or whether it suffices to adapt an existing method (method-based strategy). The step 'construct a method' is achieved by using an 'assembly-based' strategy, an 'extension-based' strategy, or a 'paradigm-based' strategy. The prior technique is based on the reuse and assembly of method fragments[1] from a certain method base [21] - called an 'assembly-based' strategy, the second technique extends a method through extension patterns, hence called 'extension-based' strategy. The last technique is appropriate when a new method must be developed by instantiating a meta-model or abstracting from a given model. This is referred to as a 'paradigm-based' strategy.

[1] Method fragments are coherent pieces of (IS development) methods [21, 62].

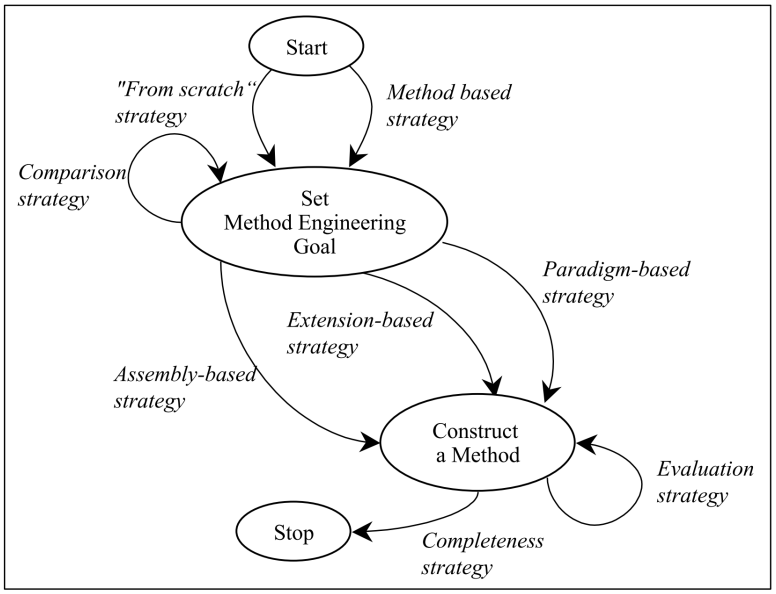


Figure 6.1: The generic process of Situational Method Engineering [125].

6.2 Method Association Approach

For this thesis project, a 'from scratch strategy' is used; since there are no similar methods that fulfill this particular goal. To 'construct a method' (Fig. 6.1), we follow the 'assembly-based' strategy: the reuse and assembly of method fragments from a certain method base [21]. As stated in Chapter 2, this project uses the *Method Association Approach* (MAA) [100] as a basis for the 'assembly-based' strategy. This approach provides concrete guidelines for assembly of the method [100]. The steps in the MAA are as follows:

1. **Identify situational factors.** Brinkkemper [21] and van de Weerd [146] demonstrated the importance of distinguishing various characteristics that identify a particular organization. These characteristics make an organization unique, and are referred to as *situational factors* in this thesis.
2. **Identify feature groups.** Luinenburg [100] described feature groups as 'sets of functional design requirements'. Feature groups consist of features: the characteristics that a method should possess or deliver. Identifying the feature groups that the CDIM method should possess allows for comparison and analysis of existing methods and selected method fragments.
3. **Select candidate methods for the identified feature groups.** Various methods are selected on the basis of their extent to which they satisfy the features needed for the CDIM method.
4. **Model relevant method fragments in a method base.** The relevant method fragments of selected candidates methods are modeled with the use of Process Deliverable Diagrams (PDDs).
5. **Associate feature groups with method fragments.** In this step, the most adequate

method fragments from the selected methods are associated with the features they should address. Luinenburg [100] suggested the association table for comparison and selection of method fragments. Method fragments are selected if they possess or contain a feature that is needed for the CDIM method.

6. **Assemble the situational method.** Using the results distilled from the association table, a preliminary version of the CDIM method is constructed, using method fragments from the method base.
7. **Validate the situational method.** This step refers to the validation of the constructed method, which is extensively discussed in Chapter 9.

There are some differences between the context in which Luinenburg and colleagues [100] conducted their study and the context of this study. Luinenburg [100] attempted to create a method to develop a (software) product, thus focusing on *what* deliverables are produced. Another study by [37] used the MAA for developing a method that focused more on the process - *how* deliverables are produced. These two different classes of methods in the MAA are called 'product-perspective' MAA versus 'process-perspective' MAA [37]. The MAA followed in this thesis is from the 'process-perspective' category. This particularly influences the way feature groups are associated with the method fragments. The following sections describe six of the seven MAA steps in detail.

6.3 Situational Factors

The first step is to identify the situational factors that characterize the environment in which the CDIM method is implemented. As discussed in Section 6.1, the CDIM will be a generic method: it is designed for a wide range of organizations. Identified situational factors of a company affect the way the CDIM method should be executed by the company. It was therefore decided that 'identifying the situational factors' should be part of the CDIM method in the form of an activity, instead of a step in the construction of the CDIM method. A company that is using the CDIM method should discuss the following list of situational factors, before they proceed. The factors affect for example which activities of the CDIM method should be executed, and which activities can be omitted. The following factors have been extracted, both from literature [23, 89, 132] and from expert interviews (Section 5.2.3). The situational factors depicted in Table 6.1 form a non-exhaustive list, and serve as a starting point for companies using the CDIM method.

The situational factors have been classified in three categories: organizational, personnel, application and development strategy. The following section briefly discusses the four categories.

Organizational. This class refers to the profile of the organization and describes the *time pressure*, *stability*, and *stakeholders*. Time pressure refers to the average demanded time to market of the organization. Stability refers to whether the company has a stable organizational environment, and whether the organization is currently or often undergoing restructuring [23]. Stakeholders refers to the number of, and access to stakeholders related to the product under consideration.

Personnel. This class refers to the characteristics of the personnel involved and describes *expertise of the software architect*, *team structure*, *team size*, and *knowledge sharing culture*.

Organizational	**Profile of the organization**
Time pressure	{low, normal, high}
Stability	{low, normal, high}
Stakeholder number	no. of stakeholders
Personnel	**Characteristics of the personnel involved**
Expertise of the architect	{low, normal, high}
Team structure	{autonomous, mixed, subservient}
Team size	{small, medium, large}
Knowledge sharing culture	{open, medium, close}
Application	**Characteristics of the application under development**
Type of software produced	{complex, simple}
Complexity of the environment	{low, normal, high}
Dynamism of the environment	{low, normal, high}
Dependency on other products	{weak, average, strong}
Risk	{low, normal, high}
Development strategy	**Characteristics of the development operations**
Development approach	{agile, waterfall, mixed, devops}
Documentation culture	{documentation driven, code criven}

Table 6.1: Situational factors, partly adopted from [89], [132] and [23], and partly distilled from the expert interviews (Sect. 6.3).

Expertise of the software architect refers to the language experience, the tool experience and the technological experience of the architect. Team structure refers to the extent to which involved teams can decide on their own tasks and responsibilities. Team size refers to the average amount of FTE within each involved team. Finally, knowledge sharing culture refers to the culture within teams, which can be affected by for instance disharmony, commitments, and performance [23].

Application. This class refers to the characteristics of the application under development and contains *type of software*, *complexity*, *dynamism*, *dependency on other products* and *risk*. Type of software refers to the complexity of the product under development, but also the backup and recovery demands and application criticality. The complexity of the environment refers to the dependency on outside suppliers, multisite development, possible involved parties, unclear requirements, or strong regulations by the government (Exp.5). Dynamism refers to the volatility of the environment, including for instance recent or often changes to the scope of the project [23]. Dependency on other products refers to the number of links to existing systems, or the number of links to future systems. Risk is determined by whether the application causes major changes in the way people work or live.

Development strategy. This class refers to the characteristics of development operations and describes *development approach* and *documentation culture*. Development approach refers to the methodology in use for a certain product, which can be agile, waterfall, devops, or anything in between. The documentation culture is determined by other situational factors but can be described as a documentation-driven culture or a code-driven culture. It is important to note

that these are extremes, and a company might be in between both extremes.

6.3.1 Profiles

Two '*profiles*' have been created as a result of Table 6.1: **systematic** and **pragmatic**. The profiles have been presented in Table 6.2. The profiles determine how the CDIM method should be executed. Companies having the *systematic* profile are companies that benefit from a systematic way of inconsistency management, while companies falling in the *pragmatic* profile benefit from ad-hoc, more practical approaches. *Systematic* companies need to be more careful skipping activities or omitting documentation, as they have for example a low knowledge sharing culture, or are developing a very complex product. The CDIM method offers guidance in which activities and steps are important for which profile. It is important to note that the profiles are indicative, they are not strict guidelines which are always supposed to be adhered to.

	Systematic	***Pragmatic***
Organizational		
Time pressure	normal/high	high
Stability	high	low/normal
Stakeholder number	large	low/medium
Personnel		
Expertise of the architect	low/normal	normal/high
Team structure	subservient	autonomous
Team size	medium/large	small/medium
Knowledge sharing culture	low	high
Application		
Type of software produced	complex	simple
Complexity of the environment	high	low
Dynamism of the environment	low	high
Dependency on other products	average/high	weak/average
Risk	high	low
Development strategy		
Development approach	waterfall/mixed	agile/devops
Documentation culture	documentation-driven	code-driven

Table 6.2: Two profiles have been created based upon the situational factors: systematic, and pragmatic. The profiles offer guidance and indication, but are not strict guidelines.

6.4 Identify Feature Groups

The feature groups form the basis for selection of candidate methods, and serve as association criteria for the construction of a method [100]. To recap, a feature group comprises one or more features, and a feature is an attribute that a method should possess, an activity that a method should provide, or an artifact that is delivered by following the method. These features are

best elicitated by means of literature study, document analyses or expert interviews [37]. The features have been identified through literature, and have been informally evaluated in the expert interviews.

Feature group	**Description**
Input of the method	This feature group contains the relevant inputs of the CDIM method.
Planning	This feature group contains features for preparation and scoping of the CDIM method.
Concern Prioritization	This class contains all the features related to prioritization of the input.
Concern Modeling	The features in this group are needed for building the matrix, and establishing a design rationale.
Inconsistency Monitoring	This feature group contains the features needed to monitor and detect inconsistencies.
Diagnosis	This class contains features related to classifying detected inconsistencies.
Handling	The features in this group relate to the strategy to handle inconsistencies.
Finalization	This class contains features to finalize the method cycle, such as monitoring and adding knowledge to the knowledge base.

Table 6.3: The feature groups that form the basis for the candidate method selection step. The table containing all features can be found in Appendix E.1.

The initial list contained a total of 39 features and was based upon a variety of literature sources [47, 59, 110, 116], especially from inconsistency management and architecture evaluation domains. After evaluation the following changes were made to the features:

- 4 features were removed;
- 16 features were added;
- the feature group *Concern Management* was renamed to *Concern Prioritization*;
- 1 feature group was added (*Concern Modeling*);
- the feature group *Concern Analysis* was renamed to *Inconsistency Monitoring*.

This changes resulted in a total of 51 features contained in 8 feature groups. A table containing all features and feature groups is presented in Appendix E.1.

6.5 Candidate Methods

The candidate methods were extracted from two domains: *inconsistency management* and *architecture evaluation*. A set of 5 criteria was used to filter out irrelevant methods. The criteria partly lean on the requirements, described in Sect. 5.3. The methods were included if they satisfied the following criteria:

1. The candidate method's applicability and validity should be demonstrated in practice (AE5).
2. The candidate method should have clear guidelines and process support (AE1).

3. The candidate method should fit in an architect's daily practices (AL3).
4. The candidate method should be based on evaluation practices.
5. The candidate method should be described in a scientific article, published to an established community of practitioners and academics.

The included methods are depicted in Table 6.4, and will be very briefly described in the following sections.

Method name	Full name	Year
SAAM	Software Architecture Analysis Method [84]	1994
ATAM	Architecture Trade-off Analysis Method [85]	2000
APTIA	Analytic Principles and Tools for the Improvement of Architectures [83]	2006
ALMA	Architecture-Level Modifiability Analysis [12]	2004
NUSEIBEH	A Framework for Managing Inconsistency [115]	2000
SAAMER	Software Architecture Analysis Method for Evolution and Reusability [101]	1997
SBAR	Scenario-Based software Architecture Reengineering [11]	1998
TARA	Tiny Architectural Review Approach [155]	2012

Table 6.4: Overview of candidate methods selected for construction of the CDIM method.

- **SAAM.** SAAM was originally developed as a means to compare competing architectural solutions [84]. SAAM is a scenario-based evaluation method that permits the comparison of architectures within the context of an organization's particular needs [84]. It is a method for determining which architecture supports the organization's needs. SAAM has been used for more than five years in over 25 evaluations of software and system architectures, in many different domains. SAAM consists of 5 steps, it's inputs are the problem description, a requirements statement and architecture description(s). One of the method's stronger properties is that stakeholders need to be involved during the evaluation. SAAM needs to be applied to a final version of the software architecture but prior to detailed design [39].
- **ATAM.** The objective of ATAM is to provide a way to determine to which extent the software architecture is capable of satisfying multiple competing quality attributes (QAs) [8]. ATAM also requires all stakeholders to be involved during evaluation rounds. ATAM is applied to a final version of the architecture. ATAM contains four phases: scenarios gathering, scenarios realization, model building and analysis, and trade-offs. ATAM is a thorough and extensive approach, it however is perceived as a real 'heavyweight' method, requiring a lot of time and resources [155].
- **APTIA.** APTIA reused ten existing techniques, to extend existing architecture evaluation methods. As an input, APTIA uses business goals, risks, and scenarios. In a case study, Kazman [83] obtained these inputs through performing an ATAM. APTIA helps architects and managers to determine which scenarios and goals they wish to focus on. It consists of the phases input gathering, architecture analysis, design approach analysis, alternatives ranking, and decision making. APTIA is relatively new, and therefore less mature than ATAM for example.
- **ALMA.** The objectives of ALMA are to predict one quality attribute, and there are three goals to test on: risk assessment, maintenance cost prediction and SA comparison. The method contains five steps and is applied at a final stage of the architecture [12]. Just

like ATAM, ALMA is a mature method, that has been validated in various projects [5]. Furthermore, ALMA requires various stakeholders to be involved, for different activities.

- **NUSEIBEH.** The framework that Nuseibeh [115] and all use for inconsistency management is not really a defined method, although it's effectiveness has been evaluated numerous times [115, 116]. The method's main activities are inconsistency monitoring, inconsistency diagnosis, inconsistency handling and impact analysis. The framework makes explicit which steps can be taken in case an inconsistency is found.
- **SAAMER.** SAAMER is an extension of SAAM with a focus on evolution and reusability. SAAMER consists of four steps: gathering information about stakeholders [101], the software architecture, and scenarios; artifacts modeling; analysis; evaluation. The activities could be carried out iteratively, which makes the method very comprehensive. SAAMER is also a mature method, evaluated in various cases [5]. Has four activities, of which one has six sub-activities. Provides a good framework structuring the process steps [5].
- **SBAR.** SBAR is a method that estimates the extent to which the designed architecture reaches the software quality requirements [39]. The method consists of four phases: define scenarios, manual execution of the scenarios, evaluating the quality attributes, and interpreting the results. One of the evaluation techniques SBAR uses is 'experience-based reasoning'. SBAR does not require involved stakeholders to be present, and is therefore more lightweight than SAAM and ATAM.
- **TARA.** The objective of TARA is to offer a simple approach for performing a basic architectural review, that is structured and repeatable, and easy to apply using limited resources. TARA is based more on expert judgment than on scenarios [155]. TARA contains seven steps: context diagram and requirements; functional and deployment views; code analysis; requirements assessments; reporting findings; creating conclusions; deliver findings. A strength of the method is that it can be used in situation where there is neither time nor resources available to perform heavyweight analyses [155].

Conclusion. Synthesizing the aforementioned information, it can be concluded that the selected methods vary greatly in objective, weight, structure, evaluation techniques, stakeholder involvement, and in- and outputs. However, this provides just the flexibility and the diversity needed to create the CDIM method, which should just be generic enough to aid software architects from various environments in managing inconsistency, but specific enough to structure the process of inconsistency management. In the next step, the selected method fragments are presented.

6.6 Model Method Fragments

In Luinenburg [100], the authors modeled **all activities and deliverables** of the selected candidate methods with the use of Process Deliverable Diagrams (PDDs). Modeling an entire method when one knows that only a subset will be used is time-consuming and ineffective. Modeling an entire candidate method is only needed when the representation of the steps and deliverables is not clear: however, the 8 selected candidate methods in this MAA all satisfied criterion 2 (*the method should have clear guidelines and process support*). Since all 8 candidate methods are very clear, only the relevant method fragments have been modeled instead of the entire method. In the standard PDD notation all relations between deliverables on the deliverable-

side should include cardinality. In this context, it was decided to omit cardinalities since they do not add value: methods are solely modeled in PDD to create a unified method base. In total, 33 method fragments have been identified from the 8 selected methods. PDDs of the 8 methods are presented in Appendix F.1.

6.7 Method Association

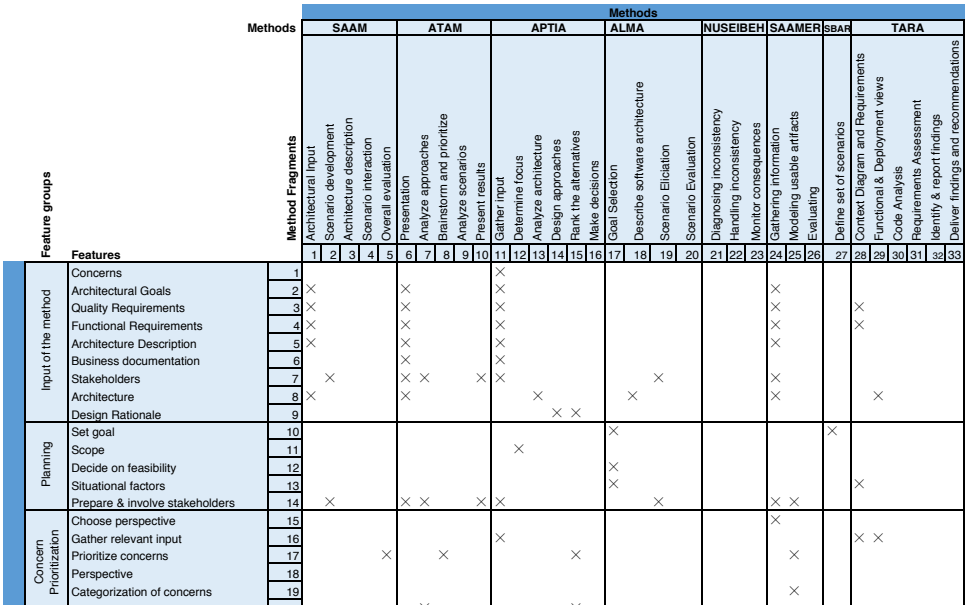


Figure 6.2: The association table (excerpt). A complete association table is included in Appendix E.2.

Method association is performed by using the 'association table' [100]. In the association table, key features of the CDIM method identified in Section 6.4, are associated with the 33 method fragments identified in the previous step. When associating features with method fragments, one can adopt two perspectives: a 'product perspective' and a 'process perspective' (see Sect. 6.2). In this research, a process-perspective is adopted, since many features focus on the steps of the new CDIM method. An association was made if the feature was present in the form of an activity or task in a candidate method. Figure 6.2 depicts an excerpt of the association table used. The table contains the 8 methods and the 33 method fragments on the horizontal axis, whereas the features and feature groups have been placed at the vertical axis. An association is represented by an '×'. It is important to note that some associations of a feature with candidate method fragments could not be performed, since some features are non-existent in the 8 candidate methods.

The association process resulted in a total of 119 associations (represented by ×'s in Fig. 6.2). In an ideal situation one would like to have as much associations as their are features. To narrow the amount of associations down, associations were prioritized on a set of of indicative criteria: method fragments that were perceived as a better fit were chosen over other method fragments; and method fragments that satisfied more other features next to the feature in question were

selected over alternative method fragments. Using the association table (Appendix E.2), the CDIM method was developed.

6.8 Summary

With the use of the Method Association Approach (MAA) we designed and assembled the CDIM method. The MAA is an assembly-based situational method engineering approach. First, situational factors were identified. Situational factors influence the time available, the involvement of participants, and the openness to the CDIM. It was decided to let the factors be a part of the CDIM method. Second, the characteristics that the CDIM method should deliver were identified from related literature. Identifying these characteristics allows for comparison and analysis of existing methods and selected method fragments. Several candidate methods were selected: SAAM, ATAM, APTIA, ALMA, framework of Nuseibeh, SAAMER, SBAR, and TARA. Based on an association matrix, method fragments - pieces of a method - were associated with the identified features. Using the results distilled from the association table, a preliminary version of the CDIM method was constructed. Chapter 7 continues this section by providing a detailed overview of all the phases and deliverables of the CDIM method.

7 CDIM method

This chapter describes the results of developing the CDIM method, based on the Method Association Approach (MAA) and advice and requirements from experts and literature. The method is called the 'Concern-Driven Inconsistency Management' method, or CDIM method in short. Section 7.1 describes preliminary principles. Section 7.3 discusses the CDIM method in brief. Section 7.4 to Section 7.10 describe the 7 activities of the CDIM method. Every step of the CDIM method is annotated with the source, which could be (a) an article, (b) a method fragment from an existing method, (c) or an expert advice. Method fragments are indicated with the name of the method (e.g. TARA), literature sources with their reference (e.g. [81]), and expert advices with their respective identifier (e.g. ICL7, or SCP2).

7.1 Method Principles

The CDIM method rests on a few principles:

- The CDIM method uses architectural- and stakeholder concerns as a basis for structured inconsistency discovery and management. It assumes that concerns relevant to the project are already identified and known.
- The method could be best seen as a series of practices that could and should (depending on the architectural process) be performed iteratively, instead of a formal sequential process that has to be executed step by step.
- The CDIM method is not a software development method. It is therefore executed in addition to the regular software development methodology an organization is using, and does not replace it.
- The CDIM method is a creative process: the focus of the CDIM method is on the result. The sequence in which the input is collected is not important.
- In one of the first activities, the architect decides whether a systematic or a pragmatic workflow is suitable for his environment. This means that some steps can be executed differently and with use of varying techniques.
- Architects are encouraged to use their preferred notation and techniques for doing steps of the method. If techniques are presented in an activity, it serves as a baseline for the systematic or the pragmatic workflow.

7.2 Concern-Driven

Several authors [67, 140] argue for concern-centric theories - approaches to architecture using concerns as first-class entities. We also adopt this view in the CDIM method, for the following reasons:

- Concerns form the starting point of each project. Without concerns, there would be no necessity for a system to be built [129, 140].
- Architecting is about satisfying concerns and demonstrating that concerns have been satisfied [95].
- Concerns form the central element in architecture and connect the problem and solution space [67]: concerns indicate important places in the architecture [95].
- By using concerns as first-class entities instead of e.g. diagrams, the architect is not constrained to a particular notation or modeling style. Models and diagram are restrictive in the sense that not every concern can be expressed in a diagram or model.
- Concerns evolve or change during a project [67]. Not regarding concerns as first class entities results in drift between changing concerns and documentation intended to address those concerns (models and diagrams). Focusing on concerns as first class entities throughout the project is therefore important [140].

Furthermore, Hilliard [67] argues that understanding of stakeholders and their concerns forms the basis for a successful architecture. Dashofy [34] argues that it is the stakeholders of a system (including but not limited to architects) that decide which design decisions are made about an architecture. The relation between concerns and design decisions is essential, and could be explained by the fact that an architecture can be seen as the result of applying a collection of design decisions, in order to satisfy concerns [8, p.32]. The aforementioned argumentation convinces us to adopt a concern-driven approach for this method. Concern-driven means that the architect collects and uses concerns as an input for the method.

7.3 Method Overview

A brief version of the method, indicating it's primary phases, is depicted in Figure 7.1. It has 2 parts: (Inconsistency Discovery and Inconsistency Management), 3 phases (Building, Using, and Solving) and 7 activities (Planning, Concern Prioritization, Concern Modeling, Concern Monitoring, Inconsistency Diagnosis, Inconsistency Handling and Finalization). Each activity contains a series of steps.

1. *Planning* consists of goal-setting and scoping.
2. In *Concern Prioritization*, the relevant concerns are gathered and prioritized.
3. Next, the identified concerns are specified in order to use them in the matrix (*Concern Modeling*).
4. During *Inconsistency Monitoring* the architect uses experiences and knowledge about the project to identify possible inconsistencies on places where concerns overlap.
5. *Inconsistency Diagnosis* takes place after the inconsistency has been detected and deals with classification of the inconsistency.
6. Next, in *Inconsistency Handling* the team settles discovered inconsistencies.

7. The last activity deals with assessing the impact, and reusing new insights on the architecture (*Finalization*).

It is advised to reiterate between the different activities. A full version of the CDIM method is presented in Fig. F.9, in Appendix F.2.

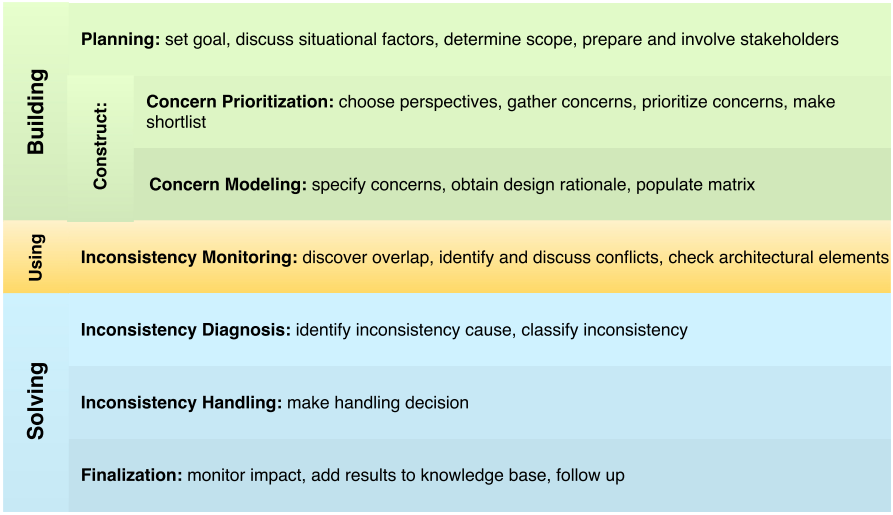


Figure 7.1: The building blocks of the CDIM method.

The illustrated building blocks and activities in Figure 7.1 are executed in a cyclic manner. In that sense, the CDIM method has a resemblance with the PDCA cycle [109]. The PDCA is a model used for continuous quality improvement and stands for *Plan Do Check Act*. The steps refer to: recognize an opportunity and plan a change, do the things you plan, check whether the results are aligned with planning, and continue by devising follow-up actions [79]. By reiterating forth and back between activities, architects can continuously update the matrix and adapt their actions. The cycle is depicted in Figure 7.2.

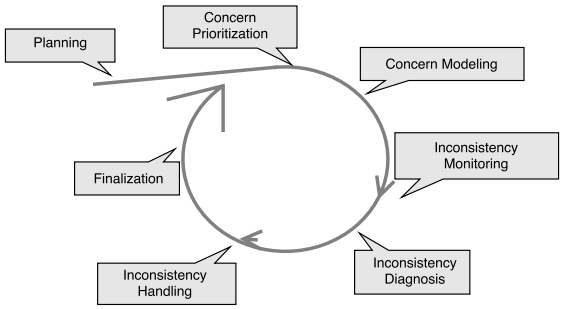


Figure 7.2: The high-level overview of a CDIM execution. It bears close resemblance to the PDCA [109].

7.3.1 Execution

Executing of the process should be done by the architect responsible for the system or product, possibly accompanied by his team of co-architects or lead developers. One person should be at least responsible for the guiding the CDIM activities. Possibly together with other people, he executes the steps. Stakeholders can be involved: if applicable, that is indicated in each step.

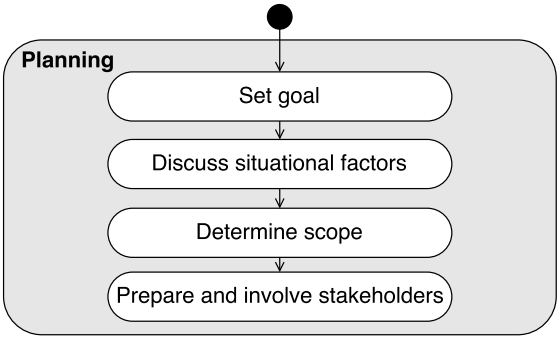


Figure 7.3: CDIM activity: Planning.

7.4 Planning

This activity is the first activity of the CDIM method, and forms the first step of the *Building* phase. It serves to determine feasibility, to prepare stakeholders to be involved (developers, users, etc.), to set a common goal and scope the required input and expected output of the CDIM method. The activity is represented in Figure 7.3.

7.4.1 Set goal (ALMA), ICL8

The first activity in CDIM method is concerned with determining the goal of the CDIM cycle. Is the product in question *already built* and implemented, or are you in an *early design* phase? This makes a difference in execution of the method.

1. **Discover and discuss existing inconsistencies.** If the architecture already exists, it is likely that the evaluation is performed with the goal to see whether the architecture contains inconsistencies that can lead to problems.
2. **Discover and analyze trade-offs.** If the architecture does not exist yet, the goal will be oriented towards risk-assessment of concerns, and discovering inconsistent assertions, assumptions, or discover and analyze trade-offs. Often, there is a need for convincing stakeholders on certain inconsistencies.

Both goals can be used at the same time. If there is no direct goal that can be identified, than there is no need for performing the CDIM method (requirement ER4).

7.4.2 Discuss situational factors (TARA, [9], SF1-7)

In essence, situational factors determine the feasibility of performing an evaluation on a particular product in a particular organization [9]. Situational factors influence the time available, the participants, and the openness to such a method [9]. In other words, they affect the way the CDIM method should be performed. As an initial fill, 14 situational factors have been developed. Note that these factors are indicative. The situational factors are described in full in Section 6.3.

	Systematic	Pragmatic
Organizational		
Time pressure	normal/high	high
Stability	high	low/normal
Stakeholder number	large	low/medium
Personnel		
Expertise of the architect	low/normal	normal/high
Team structure	subservient	autonomous
Team size	medium/large	small/medium
Knowledge sharing culture	low	high
Application		
Type of software produced	complex	simple
Complexity of the environment	high	low
Dynamism of the environment	low	high
Dependency on other products	average/high	weak/average
Risk	high	low
Development strategy		
Development approach	waterfall/mixed	agile/devops
Documentation culture	documentation-driven	code-driven

Figure 7.4: Two profiles can be adopted as a result of the contextual factors of an organization.

The goal of this step is to determine how the context of the product or organization influences the CDIM process. Based on the factors, two profiles have been constructed: *systematic* and *pragmatic*. They have been presented in Figure 7.4. Because of its environment and culture, a systematic organization needs a more structured approach, using codified knowledge [61], and a more gradual approach to architecture, using document-driven procedures. A systematic organization typically follows the CDIM method using less iterations and less cyclic than pragmatic organizations. Pragmatic organizations are on the other end of the spectrum: they might use personalization strategies for managing knowledge, using small teams with a lot of independent experts [61]. Note that the two profiles form the extremes of the spectrum, and organizations might as well fit in somewhere in the middle of the 'scale'. Where necessary and possible, steps can be tailored to a systematic or pragmatic approach. This is indicated by colored highlighted areas:

Systematic

A dark blue text box for a systematic approach.

Pragmatic

A light blue text box for a pragmatic approach.

The objective of this phase is that an architect carefully considers it's environment, in order to adapt the CDIM method to it. The architect is advised to add, adapt or remove factors if necessary.

7.4.3 Determine the scope (APTIA)

Scope of the architecture under analysis. The first question of defining a scope is to decide for which part of the architecture it is needed to detect and manage inconsistency. The architect preferably scopes the architecture on a functional basis. There are several important factors that determine the size of the scope:

- Time available;
- Documentation available;
- Complexity of the architecture.

For a complex architecture, with little documentation and time available, the scope should be minimal: may be one or two services or interfaces. A rule of thumb is that the scope should always fit on one piece of regular paper.

Scope of the input. In this step, the architect checks which documents, diagrams, business goals, and concerns are needed. *The CDIM method assumes that concerns relevant to the project are already identified and known.*

Systematic

Not all the architectural documentation, source code, technical documentation or requirements information is needed. It is effective to roughly scope the information that is needed.

Pragmatic

Pragmatic companies should discuss whether documentation is available or not.

The next aspect is to decide which stakeholder group should be involved. Identify which stakeholders should be involved during the method. Stakeholder involvement is discussed in detail in the next step. Possible stakeholder groups are described in Sect. 3.2.

Accessibility: The results of the method should be accessible for everyone in the team (requirement ER11). The architect should determine who will be able to access the deliverable, regardless of the form it is in.

7.4.4 Stakeholder Involvement (APTIA, ICA3, SCP1,2)

Resolving inconsistency may involve substantial negotiation, because inconsistency could reflect serious conflicts or trade-offs with substantial consequences [47]. Consequently, it is wise to have stakeholders involved. That does not mean that stakeholders have to actually sit in the same

room the entire time. It means that they should be made available for comments and help in detection of inconsistency by identifying overlaps.

Systematic

Consider wider ranges and larger groups of stakeholders (representatives from regulation institutions or quality authorities). Certain stakeholders are distributed across various physical locations. Involved in this case means that they are available for comments, and can influence prioritization, using video-calls for instance.

Pragmatic

Consider smaller ranges and number of involved stakeholders (users, testers, etc.). In general, the range of involved stakeholders depends on the stakeholders of the system [85]. Where it is possible, invite stakeholders to be physically present.

Decide which stakeholder groups should be involved: this is dependent on the system or product under analysis. Possible stakeholder groups are described in Section 3.2.

Output of 'Planning and scope': initial project document capturing the goal, situational factors, the scope, a clear overview of the input to be used (as a result of the goal and scope), an indication which stakeholders' involvement is needed, and a determination of who will be able to access the results. The project document is depicted in Fig. 7.5.

CDIM Project Document concerns >> **Date:** 28-02-16

Authors:	Diederik	Rene	Jan	
Involved architects:	Diederik	Paul	Rene	Jan
Version:	1.02			

Scope:

The scope has been set to the functional area of the *delivery service WEAP and the the connection with the database.*

Situational approach:

Organizational	**Application**

Figure 7.5: An excerpt of the project document that can be used with the CDIM method. The complete version is presented in Appendix G.

7.5 Concern Prioritization

This activity (see Figure 7.6) refers to gathering and prioritizing relevant concerns that will form the first-class elements of the CDIM method. **Important:** this activity and the activity 'Concern Modeling' should be executed iteratively and cyclic. The reason is that you have to know to which perspective a certain concern belongs, in order so you must partially specify each concern already. Reiteration needs to be done on the basis of architectural experience.

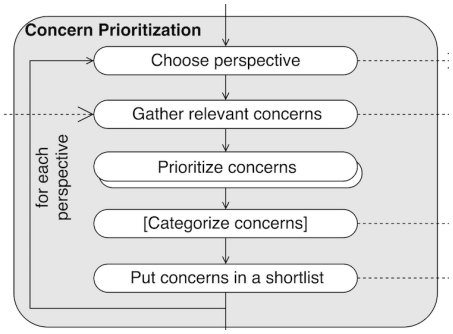


Figure 7.6: CDIM activity: Concern Prioritization.

7.5.1 Choose perspectives (SAAMER, ER8, AL1, AL2)

This step consists of choosing one or more perspectives. A perspective is *the viewing angle from which he considers the architecture and performs the analysis.* A collection of possible perspectives, from different abstraction levels, has been described in Appendix D. Note that the scope determines the perspectives that are chosen.

- If the goal is to detect inconsistencies in the architecture, one should choose perspectives where inconsistencies have the highest impact.
- If the goal is to assess risks of conflicting concerns, one should choose perspectives that are sensitive to risks.

Approaching the architecture without a certain combination of perspectives would be unstructured, very complex, and time-consuming. The introduction of perspectives leads to hierarchy (requirement ER9) and enables the architect to only include the concerns that are relevant (requirement ER8). A perspective could be any aspect, as long as it is clear to involved architects and architects. **Examples:** a functional perspective, a deployment perspective, performance (quality attribute) perspective, an information perspective, or a usage perspective.

7.5.2 Collect relevant concerns (ALMA, TARA, AL1, AL6, ER7)

From available sources of input (see 'Determine Scope'), the architect should collect concerns relevant to the perspective in question. This should be done for each of the relevant perspectives identified. **Examples:** concerns can be architectural goals, high-level requirements or demands, and constraints. The output of this step is a longlist of concerns applicable or related to a specific perspective.

Systematic

Gather concerns from the input documentation. As stated, the CDIM method assumes the concerns are already known. The task in this step is to gather the relevant concerns from available input sources. Some of the information exists in the form of Visio, PowerPoint or whiteboard sketches or more formal artifacts like UML models [155].

Pragmatic

Concerns should be gathered from input documentation, if any. In other situations, concerns should be identified from the heads of co-architects, business representatives, involved stakeholders, or other sources.

This step should be reiterated continuously, for two reasons:

- concerns change continuously, so the architect is enable to monitor and filter out less relevant or adequate concerns each time he revisits the list;
- the architect can build the matrix iteratively and incrementally (AL6) by including new concerns on each iteration.

7.5.3 Prioritize the concerns (SAAM, ATAM, ICL7,9)

Because concerns will be used to populate a matrix, the 'N^2-problem' will most certainly arise: having a set of concerns with size N, the matrix will grow to a size of $N \times N$ cells. This will be manageable for 4 concerns, but with 20 concerns, this might form a problem. Furthermore, within projects with a compact schedule or limited resources, an architect should evaluate the most critical concerns as early as possible. To prevent the 'N^2-problem' from happening, and to make sure the architect only focuses on the most relevant concerns, concern-longlists are prioritized in this step. Of course, every organization or architect has its own favorite techniques for prioritization, so architects are encouraged to use techniques to their liking. For reasons of reasons of completeness, two techniques are presented.

Systematic

Prioritization can be done by making for example 5 priority buckets: critical, very critical, critical, not critical, not worth analyzing. Each priority gets a unique weight (value) which increases per bucket. Place concerns in buckets: A concern placed in a bucket automatically gets the weight that has been assigned to that particular priority bucket.

Pragmatic

Another example of a well-established prioritization technique is card-sorting. Each participant in the prioritization is assigned a number of votes equal to 30 percent of the number of concerns, rounded up. If there are 20 concerns gathered, each participant gets 6 votes, which he can use in any way that he sees fit, 6 votes to one concern, or 6 concerns with 1 vote each [8]. In smaller groups of participants, the percentage is often increased.

Regardless of which technique is used, the priority for each concern should be clear to everyone involved. Note that prioritization should be repeated frequently, since concerns can change during a project. The output of this step is a shortlist of concerns for each perspective. **Advice:** as a rule of thumb, a shortlist of 5 concerns is advised. If the shortlist gets too long, the matrix will get unusable.

Output of 'Concern Prioritization': for each identified perspective, a shortlist of prioritized concerns that are relevant to the established goal and scope.

7.6 Concern Modeling

This phase collects all the features of the CDIM method that are needed for developing and populating the matrix, and for establishing the design rationale. The steps are represented in Figure 7.7.

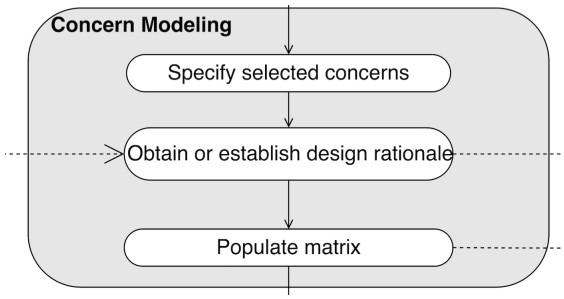


Figure 7.7: CDIM activity: Concern Modeling.

7.6.1 Specify concerns (APTIA, ER1, ER5 ER6, AL4, SCP2)

The use of templates to capture information is making methods more consistent across architects [83]. Templates provide a repeatability to the gathering and documentation of information. On the other side, an architect should not be restricted in his freedom (requirement AL4), the method should not force the architect to check or document everything (requirement ER5), and the method should be flexible (requirement ER6). Specifying concerns in a predefined template enables simplicity and consistency (requirement ER1).

The architect is advised to add, remove or customize the different items in the template if he thinks that is needed. We present two options for specifying concerns:

Systematic: a 'Concern Card'

Concerns can be specified in a 'template' using the following aspects, which makes reasoning and discussing them easier:

- ID, a unique identifier that enables the architect to reason about an individual concern.
- Name, a short textual description of a possible name of a concern.
- Description, a comprehensive definition and explanation of the concern.
- Priority, the attached priority a concern has received.
- Related stakeholders, stakeholders that are associated to the concern, or framed the concern, or have a certain interest in the concern.
- The perspective to which a concern belongs to, that enables hierarchy and categorization. perspectives enable gathering of concerns.
- Related requirements, if known, possible associated architectural requirements can be described here. These can be used during discussion.

Pragmatic

Concerns can also be specified in the form of user stories [32], which we call 'architecture stories': Example: high availability, can be specified as: "As 'service B', I need to be up and running for 99,5% of the time, for the sake of availability."

7.6.2 Obtain design rationale (ARA1, ARA2, ICA1)

In this step, a design rationale is established, in order to gain insight in how design decisions originate from the identified concerns.

Systematic

Identify and gather the documentation that contains the design rationale in the case the architecture already exists. For each selected concern, associate it with one or more design decisions and document it. In case the architecture does not exist yet, consider the potential design decisions on the basis of selected concerns, and document them carefully. For further reading consider [142] or [91].

Pragmatic

When a design rationale document is not available or non-existent, the team should devise a design rationale. In the situation where there the architecture already exists: discuss the design decisions that have been made based on the selected concerns. In case the architecture is still to be built, discuss the potential design decisions on the basis of selected concerns.

When a design rationale document is not available or non-existent, the team should devise a design rationale, discussing the effect of concerns on the current architecture. A design rationale will be used during 'Concern Monitoring'.

7.6.3 Populate Matrix (SAAMER, [9], IMO1)

In this step, an architect builds a matrix and populates the matrix with concerns. The horizontal and vertical axes of the matrix should incorporate concerns of 2 of the perspectives that the architect selected in Section 7.5.1. The perspectives determine the viewing angle from which the architect discusses the architecture. The vertical axis can contain the same perspective as the horizontal axis (`SxS`), or they can contain different perspectives (`SxA`). A matrix provides a simple visual representation of relations between concerns. The matrix maps relations between concerns of the same perspectives or concerns of different perspectives.

Systematic

Architects using a systematic approach should draft the matrix in the spreadsheet provided with the CDIM method, or use the tool that comes with the CDIM method[a].

[a]currently, the tool is still in a conceptual phase. Development of a real usable tool is a topic for future research.

Pragmatic

Architects following a pragmatic approach may even draft a matrix on a whiteboard. To specify the concerns, the architect could use colored note-it's. In that case, the matrix it self is not very reusable, however the note-it's can be reused easily.

Output of 'Concern Modeling': this phase results in more detailed and specified concerns, a matrix populated with the selected perspectives and concerns, and their associated relevant design decisions.

7.7 Inconsistency Monitoring

The aforementioned steps were all part of the 'Building' phase, and focused on building the matrix and populating it with relevant concerns. 'Inconsistency Monitoring' forms the 'Using' phase, since the architect uses the matrix to discover and discuss possible inconsistencies. This step will structure the process of checking the architecture for inconsistencies. This phase relies on the expertise of the architect: expert opinion is typically required to determine whether architectures are satisfactory or not [83]. Most architecture-based methods rely on expertise. Given the deliberate simplicity of the CDIM method, and the complexity of a software architecture context, the steps in this phase are inevitably one of judgment rather than quantifiable assessment. The steps are represented in Figure 7.8.

Remark about the matrix

By using concerns as first-class citizens in the matrix, the architect checks the relations between the concerns to determine where to look for inconsistencies. The matrix does not aim for prescribing *how* inconsistencies should be checked. It aims to present the architect with structured guidance on *where* inconsistencies may arise. The matrix enables the architect to align concerns from two perspectives on the two axes of the matrix. The principal idea behind the matrix is that it provides an overview of how concerns from two perspectives are overlapping or conflicting. The cells refer to '*hotspots*' - areas in the architecture where concerns could overlap or conflict. Subsequently an architect may check the contents of a cell to see where inconsistencies arise in design decisions, models, etc. Each cell can contain one or more instances (overlaps, conflicts, or inconsistencies). The matrix is depicted in Figure C.4 (in Appendix C). **Important:** The way of working with the matrix differs for the two profiles.

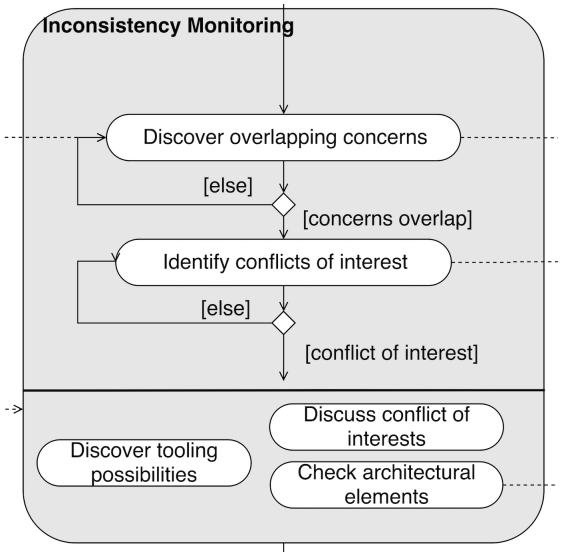


Figure 7.8: CDIM activity: Inconsistency Monitoring.

Systematic

With a systematic profile, it is advised to execute each step for all cells of the matrix, and then move on to the next step. This goes for all the steps in 'Concern Monitoring'. Thus, the architect first searches for overlaps in the cells, and subsequently searches for conflicts in all relevant cells. Next, for all cells, he discusses the conflicts and checks in the architecture. We refer to this as the **horizontal approach**: do a relevant step for all cells, and then move on the the next step.

Pragmatic

In this case, the architect is advised to browse through a cell, and subsequently execute 'identify overlapping concerns', 'identify conflicts of interest', 'discuss conflict of interest', 'check architectural elements', and all further steps up to 'Finalization'. These steps can be executed quite fast, and result in an immediate handling decision when an inconsistency has been found. We refer to this as the **vertical approach**: concentrate on a single cell and execute the relevant steps.

7.7.1 Identify overlapping concerns (SAAM, SAAMER)

The first step in this phase is to identify the concerns that have overlap. For the concerns in question, use the *design rationale* (obtained in 'Concern Modeling') to identify:

- which design decisions have been made in the case the architecture is already developed;
- which design decisions should be made in case the architecture is to be built;

The design rationale is thus used to identify whether concerns have interacting design decisions. Different unrelated concerns may necessitate changes or decisions on the same elements in the architecture. In such situations, concerns interact - have a certain overlap. Determining the interaction of concerns leads to insight in whether the architecture supports an appropriate separation of concerns. **Overlap:** concerns have an overlap if their associated design decisions influence each other. This can be both a negative influence as a positive influence. Two concerns can have no overlap. Some concerns do however influence each other. For example a concern about availability is affected by a concern that states that data may only be stored in one place. The overlap between these two concerns is that there is a single point of failure, if data is not stored redundantly. The output of this phase are identified overlaps between two or more concerns.

Systematic

In systematic approaches, a design rationale is usually documented explicitly. In this case, identifying overlap is an activity aimed at the the information in the design rationale.

Pragmatic

In pragmatic approaches, the design rationale may be less explicitly documented. In this case, identifying overlaps is an activity aimed at discussion and extracting the knowledge from other evaluating architects.

7.7.2 Identify conflicts (SAAM, SAAMER)

Overlapping concerns can represent conflicts of interest in the architecture. Conflicting concerns could have inconsistent specifications or design decisions. **Conflict:** concerns are in conflict if their associated design decisions are mutually incompatible; or negatively affect each other.

Questions that facilitate finding conflicting concerns are:

- Are the concerns functional or non-functional?
- Are assumptions about both concerns and their implications aligned?
- Are all design design decision documented?
- Are all the design decisions known?
- Are the concerns framed by different stakeholders?
- Are the implementations made by different teams or roles (advice ICA4)?

If a conflict of interest is found, it is necessary to address, identify and mark it.

7.7.3 Discover tooling possibilities

In this step, the architect assesses possible if it is possible to check the architecture upon the identified conflicts with the help of tooling. A lot of low-level inconsistencies are detectable with the use of ready-made tools that can be executed upon source code or available models.

7.7.4 Discuss conflict (APTIA, IMO1,6, ARA1,2)

In this step, certain 'hot spots' (places where there are conflicts of interest among concerns) are discussed by the architect and stakeholders. It is the starting point of a dialogue whether a conflict of interest has been settled without any inconsistencies in implemented code, models, or diagrams.

Role involved stakeholders: significant or large conflicts should be discussed with related stakeholders. Involve the stakeholders from which concerns originated. Stakeholders were already informed in 'Planning', and can now be involved by means of physical or digital interaction. Their role is to aid the architect with deciding on how conflicts affect the architecture, and which conflicts could be problematic. This should always be done in line with the project goal.

Some initial questions that an architect should ask:

- What elements should be checked?
- Which questions can we ask to verify this?
- How should this be accounted for in the architecture?

The output of this step are clear indications of which conflicts:

- have inconsistent models, diagrams, or code if the architecture is already developed;
- have potential inconsistent design decisions, if the architecture is not developed yet.

.

7.7.5 Check architectural elements (TARA, IMA3)

In this step the architect checks architectural elements upon identified conflicts of interest, with the goal to find inconsistent elements or specifications. Checking the architecture for inconsistent elements should be performed by using reasoning based on logical arguments, rather than explicit rules to check elements. The drawback is that this approach is less explicit and more based on subjective factors as intuition and experience. The advantage is that this is easier and more flexible [155]. The output of this step is a list of the most essential identified inconsistencies in the architecture.

Revisit 'Concern Prioritization':

Concerns may evolve or change during a project (advice IMA1). Therefore, in this step the architect is advised to reiterate back to 'Concern Prioritization' in order to revisit the selected concerns if needed.

Output of 'Concern Monitoring': the output of the Inconsistency Monitoring phase are the identified overlaps, conflicts of interest and possible inconsistencies, which are recorded in the matrix. In case of a systematic approach, the architect will move on to the next phase with a list of found inconsistencies. In case of a pragmatic approach, the output of 'Concern Monitoring' is a single inconsistency that will be classified and handled in the following sections.

7.8 Inconsistency Diagnosis

The second part of the CDIM method (*Inconsistency Management*) targets to support the architect with governing and handling inconsistency. The first activity ('Inconsistency Diagnosis') of this part focuses on determining what elements are inconsistent, what the cause is, and what the classification should be.

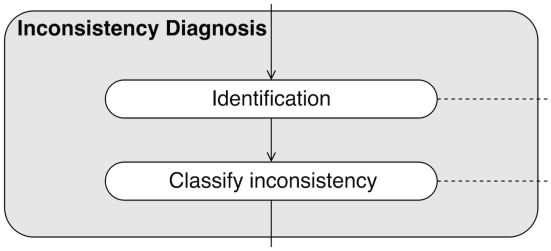


Figure 7.9: CDIM activity: Inconsistency Diagnosis.

7.8.1 Identification (NUSEIBEH, ICH1,2,3)

After an inconsistency has been found, the architect assesses the inconsistency to determine what kind of inconsistency has been found, and what the cause of the inconsistency is. There are several forms of inconsistency in software architecture, and one type is more easily solved than another. A detailed overview of inconsistency can be found in Chapter 4. Furthermore, the team should focus on the cause of the inconsistency. Examples of causes of inconsistency could be:

- Contradicting assumptions, limited knowledge about other actor's knowledge.
- Incompleteness of descriptions, contradiction between requirements.
- Incompleteness of models, contradiction between models, architecture documentation not kept up to date with the system, multiple specifications, multiple actors, multilingualism.
- Uncontrolled evolution of code, different tools, different actors.

Having an up-to-date design rationale can be beneficial in finding root causes [142], and diagnose impacts. Create a clear conceptual image of an inconsistency for every relevant stakeholder, using the techniques that are found appropriate.

Role involved stakeholders: in this step, stakeholders may be involved to help identifying the cause of inconsistencies. For example: an involved developer may know better how two diagrams could have become inconsistent than an architect, because he worked on both diagrams and saw them become inconsistent over time.

7.8.2 Classify inconsistency (NUSEIBEH, APTIA, IMA1, IMO3, ICL1, 2, 4, 5, 6)

Classification of inconsistencies is a crucial step, because it determines how to proceed and handle the inconsistency. Each inconsistency is given a classification, which helps the architect deciding what to do with an inconsistency.

This section presents four aspects which could be considered to classify an inconsistency: impact/risk, business value, engineering effort, and characteristics. Figure 7.10 presents the four

classes and several factors that could be considered when classifying an inconsistency. These factors are based upon results of the expert interviews.

Systematic

Answering each factor leads to a certain score, and based on the score of each factor a score is 'calculated' for a category. The score for a category partly determines the follow-up action in the *Inconsistency Handling* phase.

Pragmatic

Each factor serves as a guide. An answer to each question is necessary, assigning a score to each question is not.

Impact/risk		
Are there direct current consequences?	Yes = 2, No = 0	Low impact — High impact; 0 — 8
What are the costs/risks of ignoring?	High = 2, Med =1, Low = 0	
Are there any indirect consequences?	Yes = 2, No = 0	
Is there a negative impact after solving?	Yes = 1, No = 0	
Will it happen again?	Yes = 1, No = 0	
Business Value		
Is there an important business rationale?	Yes = 4, No = 0	Low business value — High business value; 0 — 7
Important stakeholders involved?	Yes = 3, No = 0	
Engineering effort		
Are there any design alternatives?	Yes = 0, No = 2	Low effort — High effort; 0 — 7
What is the required effort of solving?	High = 0, Med = 1, Low = 2	
What is the required effort/rework of alternative solutions?	High = 0, Med = 1, Low = 2	
Do we have rulesets, tools or checks available to solve this?	Yes = 0, No = 1	
Characteristics		
What is the cause of the inconsistency?	n.a.	[Score is dependent on factors]
What is the type of inconsistency?	n.a.	

Figure 7.10: An initial taxonomy for classification of inconsistency.

Risk. The category 'Impact/risk' is a essential, as risk and cost always have been important drivers in architecture [81]. Impact refers to the urgency and the scope of the consequences an inconsistency has on the architecture and the system. Factors within the category 'Impact/risk' address issues that focus on the necessity of solving an inconsistency right away: if there are direct current consequences, the inconsistency has a high impact. However, if there are no direct consequences, but there is a high risk or cost of ignoring the inconsistency, the inconsistency still could have a large impact on the architecture.

Business value. The 'Business Value' category focuses on organizational factors such as whether an inconsistency is related to or caused by important stakeholders.

Engineering effort. 'Engineering effort' focuses on whether solving an inconsistency will require a lot of effort, and whether exploring the design alternatives is of added value. As one expert mentioned: "*classify inconsistencies on the basis of how much work is required to improve or repair it.*"

Characteristics. The fourth category focuses on the characteristics of the inconsistency itself. These vary in each particular situation, however, these should be considered as well [116]. Scores are omitted on purpose, as the factors are rather subjective.

It is important to note that the factors within each category are indicative. An architect is advise to substitute or improve factors if that is necessary. By going structurally through a series of factors, it is assumed that the architect is supported in the mental process of classifying an inconsistency.

Output of 'Inconsistency Diagnosis': inconsistencies with a type, cause and a classification in terms of a 4-tuple {high,high,low,low}.

7.9 Inconsistency Handling

The next phase guides the architect in deciding on a follow-up action for an inconsistency. As an input, it uses the classification established in the previous step. **Important note:** in some contexts, it is advised to shortly review the concerns as they might change over time.

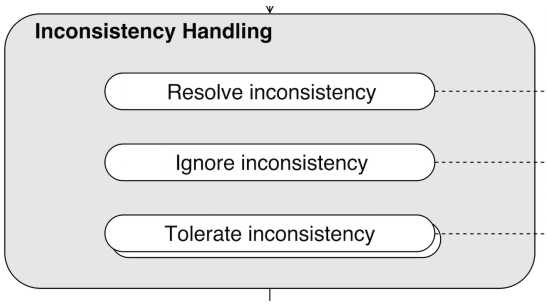


Figure 7.11: CDIM activity: Inconsistency Handling.

7.9.1 Decide upon follow-up (IMA1, ICH1,2)

Dependent on the output of the classification phase an architect has to make a decision on whether to resolve the inconsistency or not. The 5 handling decisions 'Resolve', 'Ignore', 'Postpone', 'Bypass', and 'Improve' are presented in Figure 7.12 and are discussed in the following sections. They are based on the work of [115, 116] and the advices IMA1, ICH1 and ICH2. It is important to note that the values of impact, business value and engineering effort that lead to handling actions are not restrictive. Handling an inconsistency is always context-specific and requires human insight.

Role involved stakeholders: if resolving an inconsistency relies on changing or making important design decision, the relevant stakeholders should be addressed. Their role is to aid the

architect in deciding upon important questions.

Resolve	Ignore	Postpone	Bypass	Improve
Impact/risk: high Business value: high Engineering effort: -	Impact/risk: low Business value: low Engineering effort: -	Impact/risk: med/low Business value: med/low Engineering effort: high	Impact/risk: low Business value: med/high Engineering effort: high	Impact/risk: low Business value: high Engineering effort: low

Figure 7.12: The handling decisions and their respective classification values for an inconsistency.

Resolving the inconsistency is recommended if the impact is high and the business value is high, regardless of the engineering effort. Inconsistencies that have a high impact and a high business value should have the highest priority. Resolving the inconsistency could be relatively simple (adding or deleting information from a description or view, or updating certain code), possibly using tools (IMO4,5). However, in some cases resolving the inconsistency relies on making important design decisions (e.g. the introduction of a complete new database management technology). In that case it is advise to discuss the question by involving the stakeholders again. Immediate resolution might not be the preferred option, because there is no time nor the right people to make a decision currently. In that case, one could follow a different strategy such as ignoring, postponing, bypassing or improving it.

Ignoring the inconsistency is recommended if the potential impact is low, the business value is low, and the engineering effort is high. When the effort to solve an inconsistency is large compared to the risk that the inconsistency will have adverse consequences, ignoring the inconsistency is an adequate step [115]. Best practices have shown that it is wise to revisit such decision when a system involves and in certain moments in time [115].

Tolerate. **Postponing**, bypassing and improving are strategies in which the architect in essence *tolerates* the inconsistency. In some cases, *postponing* - or deferring - the decision to resolve is recommended when both impact and business value are relatively low, and engineering effort is relatively high. In such cases, postponing provides the architect with more time to elicit further information [115] or meet other stakeholders before deciding whether to resolve or to perform other actions. Nuseibeh [115] points out that it has proven to be useful to 'flag' the affected parts.

Bypassing is a strategy in which the resolve-decision is circumvented by adapting the architecture or the description in such a way that the inconsistency itself still exists, only is bypassed. This means that they come up with a solution that not directly solves the inconsistency, but will work for their part. It is important that involved and affected stakeholders are informed. It could be using an other diagram instead of the inconsistent specification, or using another consistent library, or service instead of an inconsistent one. With regards to the classification of the inconsistency, bypassing is recommended when the current impact is low, but the business value and the engineering effort are both relatively high. Because the business value is high, continuity is important and so bypassing is an adequate solution.

In other situations, it might be cost-effective to **improve** an inconsistency without necessarily resolving it. This handling approach refers to strategies that help the team improving the areas where inconsistencies have been found. When time pressure is high, and risk is low, this might be a good alternative. To improve an inconsistency, a document that contains inconsistent diagram

can be annotated with text or explanations with the right instructions, in order to alleviate possible negative or costly consequences of the inconsistent specification.

Output of 'Inconsistency Handling': this phase leads to a handling decision, possibly in the form of a request for change or a resolution, or in the form of ignorance or toleration.

7.10 Finalization

This is the final practice of the method, in which the team finalizes the work, the focus is on reusing gained insights.

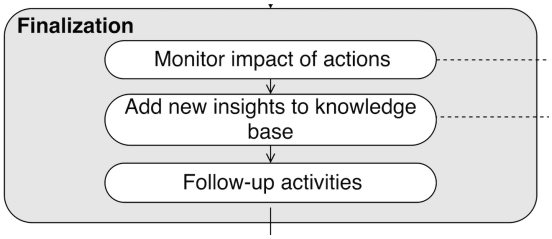


Figure 7.13: CDIM activity: Finalization.

7.10.1 Monitor impact of actions (ALMA, NUSEIBEH)

Regardless of which action is taken, it will have impact on the architecture. It is therefore crucial to monitor the consequences of the actions taken, and identify elements are affected. This requires knowledge on which elements change. If the architect solves a certain inconsistency, this might be of influence on other concerns. The architect should monitor impact by at least addressing the following actions:

- Assess whether the handling action intervenes with other handling actions;
- Assess whether the handling action affects existing concerns: go back to 'Concern Prioritization';
- Analyze if the handling action results in new concerns.

Undiscovered relations between elements can result in unforeseen ripple-effects when actions are performed: it is therefore not always possible to determine the full impact of actions. Next to architectural elements, a change can also affect stakeholders, or have an impact on identified concerns. It is important to frequently revisit the concerns. When needed, the architect can carefully intervene or adjust certain actions.

7.10.2 Add new insights to knowledge base (AE6)

A knowledge base forms the starting point for the next CDIM cycle, as reusability is considered as an important asset that a method should possess. What is added to the knowledge base is dependent on the context at hand. Examples of input to the knowledge base are new insights, identified inconsistencies, strategies for handling inconsistencies, shortlists of concerns, used matrices, and so forth. Enabling accessibility of the output of the CDIM method is recommended

(requirement RE11). The CDIM method is believed to be of real added value when the outputs 1) enable discussions among team members and 2) serve as a reference for keeping the architecture consistent.

7.10.3 Follow-up activities

In this step, the team focuses on whether the established goal has been completed and whether their insight in the architecture has improved. As a follow up, the team can decide to organize knowledge sessions or the contents of the knowledge base as a kind of wiki, in order to foster communication between teams.

Output of 'Finalization': additions to a knowledge base in the form of detected and handled inconsistencies.

7.11 Summary

This chapter aimed to describe the 'Concern-Driven Inconsistency Management' (CDIM) method, as it resulted from the MAA and expert interviews. It consists of 3 phases (Building, Using and Solving), which are divided in 7 activities (Planning, Concern Prioritization, Concern Modeling, Concern Monitoring, consistency Diagnosis, Inconsistency Handling and Finalization). In these 7 activities, the evaluating architect studies the architecture of the system in question with the goal to find inconsistencies or conflicting design decisions. Using concerns as an input to the CDIM cycle, the architect constructs a matrix, which aids him and the involved stakeholders to identify overlaps and discuss inconsistencies. Inconsistencies that are detected during the cycle can be diagnosed and handled, using the detailed guidelines also present in the CDIM method. The CDIM method's last activity enables reuse and helps the architect with the follow-up. The two profiles that can be followed during the method enable the architect to adopt the method in different contexts.

8 Evaluation

The CDIM method will be evaluated through lightweight case studies and expert interviews, focusing on the *requirements satisfaction* (identified in Chapter 5), the *perceived usefulness*, the *strengths* and *weaknesses* and the intention to use the method. These constructs form a recurring pattern in the discussion of the results for each case study and interview.

8.1 Method

Based on the three evaluation criteria, the CDIM method is evaluated qualitatively. Evaluation and data collection was performed through performing two lightweight case studies and two semi-structured expert interviews.

8.1.1 Lightweight case studies

Two lightweight case studies have been executed. As a preparation on the lightweight case study, both experts were asked in advance to think of a small part of the architecture of a recent or current project. Furthermore, the participants were asked to choose and collect a number of concerns that are relevant to that specific part of the architecture. The lightweight case study consisted of documentation study beforehand, the execution of a single instance of the CDIM method, and documentation study afterwards. The instantiation consisted of four parts: explaining the steps and deliverables of the CDIM method; discussing and illustrating the CDIM tool that has been developed; and performing a limited instance of the CDIM method. In this situation, limited refers to a limited amount of concerns, a virtual involvement of stakeholders, and a small part of the architecture of a real system. From here on, participants in the case studies are referred to as 'case experts'.

8.1.2 Expert interviews

In the second part of the evaluation, the CDIM method and its deliverables were presented to two expert software architects. An in-depth walkthrough of the CDIM method was performed

together with an illustration of the screens that have been presented in Appendix C. We refer to participants of the expert interviews as 'interview experts'.

After both the case studies and the expert interviews, the participants were presented several statements on the *requirements satisfactions*, and various questions on the *perceived usefulness*, *strengths and weaknesses* and *intention to use*. Transcripts of the case studies and interviews were produced and sent to the participants on the same day or the day after. Further questions were discussed through e-mail.

8.1.3 Participating experts

The participating experts in the evaluation were the same experts that participated in the construction and development of the method. We recognize this is a limitation (see Sect. 10.1 for a discussion). The rationale for inviting the same experts is that they already were familiar with the topic, which saved time in the introduction and explanation in each session. Their information has been summarized in Table 8.1. 'Years of exp.' stands for years of experience as a software architect, 'Org. type' stands for the type of organization, 'Org. size' stands for the size of the organization, and 'type' stands for the type of evaluation session that was held with the expert.

#	Type	Org. size	Org. type	Job title	Years of exp.
1	Lightweight case study	Large (2.000+)	IT Services	Principal IT architect	20 years
2	Lightweight case study	Large (700+)	Online retail	Lead IT architect	07 years
3	Expert Interview	Large (2.000+)	IT Services	Software architect	25 years
4	Expert Interview	Small (50+)	Healthcare IT	Software architect	10 years

Table 8.1: The background information on participating experts of the evaluation.

8.1.4 Questions and statements

Requirements satisfaction. 12 statements were presented to the experts participating in the case studies and interviews, in order to assess the satisfaction of 10 out of 21 identified requirements. Table 8.3 shows how the requirements are covered by the statements. The coverage of the other 11 requirements is described in Section 9.1. The experts were asked whether they agreed with the statement or not, using a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree, 3 = neutral). It has been established that there is no real difference between a 5-point and 7-point scale [36], so it was decided to use a 5-point scale for simplicity. The 12 statements are included in Table 8.2. Three statements were formulated negatively in order to avoid inattentiveness in the responses [104]. These statements are marked with a black downwards triangle.

Perceived usefulness. Questions on this evaluation criterion aim to identify the perceived usefulness of the individual components of the CDIM method, such as the situational factors, the guidelines for inconsistency diagnosis and the offered technique for making a handling decision. Furthermore, the participant is asked if he has an intention to use the method, and whether there are other contexts or purposes in which the CDIM method could be useful.

	#	Statement
	1	The guidelines and process support is clear.
	2	It is clear to me in which stage the method should be applied.
	3	I have a technique for each step if it is required.
▼	4	It is not feasible to identify and involve stakeholders in this process.
	5	A design rationale and a knowledge base are effective means to promote reusability in this process.
	6	The method fits in my daily practices as a software architect.
▼	7	The method feels restrictive and compelling.
	8	The method does not constrain my architectural freedom.
▼	9	The method is difficult.
	10	The method is flexible.
	11	Concerns are an effective way to detect inconsistency.
	12	Using a matrix is an effective way of detecting overlaps.

Table 8.2: The statements presented to experts after each session.

Requirement from:		Addressed by:
Architecture Evaluation		
AE1	Guidelines and process support.	Statement 1
AE2	Approaches.	Statements 2,3
AE3	Stakeholder Involvement.	Statement 4
AE6	Reusability.	Statement 5
Architecture Description Languages		
AL1	Support modeling of specific sets of concerns.	Statement 11
AL3	Fit in an architect's daily practices.	Statement 6
AL4	Should not constrain the architectural freedom.	Statement 7,8
Interviews		
ER1	Simple and consistent in use.	Statement 9
ER5	Should not force one to check and document everything.	Statement 7,8
ER6	The method should be flexible.	Statement 10

Table 8.3: In total 10 requirements are covered by statements presented to the experts.

Strengths and weaknesses. Questions in this category aimed to assess whether the experts, that participated in the case studies, where satisfied with the outcomes of the CDIM method, and whether they thought they gained more insight in the architecture. Participants that performed the expert interviews were asked to do this on the basis of their expertise as an architect. In addition, experts were asked what the perceived benefits and obstacles of (applying) the CDIM method are, and whether they experienced difficulties with the quantity of the information (which refers to requirement ER8). Lastly, the interviewees were asked to come up with possible enhancements or improvements of the method. The questions can be found in Appendix H.4.

8.1.5 Prototype

The prototype tool constructed in Appendix C is not explicitly evaluated during the sessions. This was a result of two factors: (1) the evaluation of the conceptual prototype could restrict the available time remaining for evaluation of the CDIM method, and (2) the design of the conceptual prototype is not validated, thus could affect the outcomes of the evaluation and present

participants with a biased representation of the CDIM method.

8.2 Results of the Lightweight Case Studies (Experts 1 & 2)

This section presents the uninterpreted and raw results of both lightweight case studies.

8.2.1 Case Study 1 (Expert 1)

The first case study has been executed at the company of Expert 1, an IT services organization. The description of the context, case, and execution of the CDIM method are presented in Appendix A.1 for reasons of brevity. The instantiation of the CDIM method led to various comments, questions and discussion points. The results are categorized using the constructs discussed in Sect. 8.1.4.

Requirements satisfaction

Overall, the participant was positive towards the statements presented. However, in his opinion some aspects needed improvement. He summarizes this nicely:

> "*I think I am getting enthusiastic! I see the point of what you are doing here. Prioritizing those concerns, weighing up the concerns to other concerns, cool! I am still curious how you want to embed this.*"

	Statement	Rating	Comments (case study 1)
	1	3	Steps are clear. Try to include explicitly what happens in the matrix' cells.
	2	5	The phase in which you can apply the CDIM is clear. It can be applied continuously, when concerns pop up or requirements emerge.
	3	5	Every architect is capable of prioritizing and classification.
▼	4	1	It is always desirable, not always feasible: that is dependent on the context of the project.
	5	3	I haven't experienced this well enough to provide a good answer.
	6	4	It is partly one of my key activities as an architect, however the CDIM formalizes the steps that are implicitly taking place in my head.
▼	7	2	-
	8	5	-
▼	9	3	The method is not difficult. However, there are some unclarities in the method. During the case study we noticed that there was discussion regarding the definition of a concern. Present architects with the freedom to define concerns themself.
	10	4	-
	11	4	The concerns are effective in creating awareness, regarding to inconsistency. However, this means that you have to have discovered them, so yes, indeed, the method enables you to discover the inconsistencies you have not yet discovered.
	12	4	The matrix is an effective way to discover inconsistency, however, it has a limitation: it is two-dimensional. Each decision you make affects existing concerns, so a direct feedback loop into the matrix would be added value.

Table 8.4: Responses of expert 1.

The individual scores on the statements are presented in Table 8.4. Overall, the participant rated the statements positively. The expert has some critique on the guidelines of the CDIM method. A strength of the method was stakeholder involvement, although not always possible, as stated by the expert. The most important findings are presented below:

1. **Finding: the CDIM method is flexible and easy.** In general, the method was perceived as a flexible, easy approach towards structuring the considerations that an architect often makes implicitly. Statements 9 and 10.
2. **Finding: the CDIM 'forces' the architect to deal with inconsistency management consciously.** Statement 6. "*It is partly one of my key activities, however the CDIM formalizes the steps that are implicitly taking place in my head.*" The CDIM method structures the thinking process that an architect often follows implicitly.
3. **Finding: the matrix is an effective way to discover inconsistency.** Statement 12. "*If you have collected the right concerns, it is super effective to put them in a matrix, to browse through, and discuss them.*" The limitation is that the matrix is 2D: as a result, the architect can only compare two perspectives at the same time. "*People are used to two-dimensional matrices, and a matrix will always be a good way to discover these conflicts, so I would stick with the matrix.*"

Perceived usefulness

The CDIM method's perceived usefulness was assessed by statements 11 and 12 and various questions about the individual steps of the method. The situational factors - in order to help an architect decide how to approach the CDIM method - were indicated to be a useful means, but the participant could not verify the quality of the individual factors.

4. **Finding: the CDIM method is useful as a guideline during a stakeholder conversation.** "*You can use the matrix for showing an involved stakeholder that you have thought of a solution very well and systematically. It really structures the conversation*".
5. **Finding: classification of inconsistencies (diagnosis) is useful.** "*The constructs and questions were quite helpful for structuring the classification process. However, those numbers are rather arbitrary in my opinion. As an architect, you have to answer those questions, but not really assign a value to each question.*"

Strengths and weaknesses

The participant indicated to have an intention to use the CDIM method, although he stated that he would only use the parts that he needed. A key strength he identified was:

6. **Finding: the CDIM method structures and explicates the inconsistency management process.** An important strength of the method was that it "*really starts a discussion*". The increased structure and explicitness of the process of inconsistency management was found to be the major strength.

Furthermore, the expert indicated possible weaknesses:

7. **Finding: the CDIM method is not applicable if concerns are defined on the wrong level.** "*So that can be seen as an obstacle, how can you define the concerns on such a level that the matrix actually yields fruitful results. Formulating concerns is difficult, especially because many of them are implicit. Making them explicit results in the fact that the architect has to reason about them in a certain cell, while he may prefer analyzing the concerns implicitly.*"
8. **Finding: the contents of the matrix' cells must be improved.** "*For every solution, there are alternatives. However, every alternative should be validated against concerns. I would specify in the matrix not only the type of handling decision you make, but also which solution has been proposed.*" The participant mentioned that you could decide to postpone an inconsistency, but record two possible solving strategies in the cell, for later inspection.
9. **Finding: the CDIM method should include an initial filled matrix, with example concerns on different perspectives.** The participant suggested an initial fill of the matrix, to start off the discussion and as an example of which level of detail is needed to let the matrix be effective. "*The CDIM should come with an initial fill, so you start the discussion on several concerns!*"
10. **Finding: the impact of a handling action should be explicitly monitored.** "*It would be useful if the CDIM method offered the opportunity to show how the chosen handling decision possibly interacts with the other concerns. Ideally, the output of the method should be the input for a new cycle of the method.*" "*If you solve a certain inconsistency, this might be of influence on other concerns. Clearly and explicitly guide the architect to check this.*"

8.2.2 Case study 2 (Expert (2))

The second case was executed at a webshop. The description of the context, case and execution of the CDIM method are presented in Appendix A.2. As a result of the case study, several comments and remarks arose. These are the topic of following sections.

Requirements satisfaction

In general, the participant was quite positive towards the statements presented afterwards. The architect indicated that the distinction between inconsistency discovery and inconsistency management was a good choice. Several key findings are discussed below:

1. **Finding: the CDIM method provides structure without constraining the architect.** The toolkit approach and flexibility of the method were received positively, and the CDIM method provided structure without constraining the architect. Statement 8.
2. **Finding: stakeholders should always be involved in inconsistency management.** "*You ideally want to involve many stakeholders as soon as possible, to avoid "Ivory Tower*[1]*"-stories.*"
3. **Finding: concerns are difficult concepts to work with.** "*I have my doubts on using concerns to do this. I really had to put all my effort in devising a name for the concerns that we identified. I really had to challenge myself: 'how can I express these conflicts in terms of concerns?'. I know that I want to discuss thát particular cell in the matrix, but I*

[1] see Kruchten [92] for explanation of architectural anti-patterns - things that an architect shouldn't do. Ivory tower refers to the architecture team being isolated from developers, therefore creating strong misfits and contempts.

	Statement	Rating	Comments (case study 2)
	1	4	Clearly explain the stakeholder roles.
	2	5	-
	3	4	Add guidelines on what elements are and are not allowed in the matrix.
▼	4	1	Prevent 'Ivory Tower' kind of stories. Involve as early and as many different as possible.
	5	3	Knowledge base is useful, design rationale is often implicitly available.
	6	4	Not every step, however I find this inconsistency management phase useful.
▼	7	1	Because we are solving-centric, the inconsistency discovery part feels a little bit restrictive. But that is context-dependent.
	8	4	Within our context, you already are doing a few of this steps, only now you make them explicit, which is supportive.
▼	9	2	No, but have a look at the definition of 'concerns'.
	10	4	Because of the toolkit approach I have the feeling this method is flexible.
	11	3	I found it difficult to define appropriate names for concerns, while I already was working on solutions for those conflicts.
	12	5	Using a matrix is very effective, if you are able to collect the right concerns. A template example matrix would be great for architects.

Table 8.5: Responses of expert 2.

found it difficult to express that in terms of concerns in those axes. That was a bit difficult. It is not necessarily good or bad, but I am not sure this is the most effective way."

4. **Finding: concerns need to be properly defined before they can be used in the CDIM method.** "*During the execution of the method we already had some discussion on what exactly is the definition of concern and what exactly is a perspective. I am convinced that the architect should decide - for his own organization and environment - what concerns and perspectives entail. The architect needs to have the freedom of deciding on what is a concern and what is not.*"

In addition, he indicated that the method in its entirety may not always be applicable to his current situation. According to him, a lot of steps of the CDIM method are performed implicitly in their organization, and while the CDIM method would definitely be useful in making the implicit steps explicit, the focus in his company would be more solution-centric. He argued that inconsistency discovery felt a little compelling, as he would more focus on inconsistency management. He mentioned that a lot of the steps are currently performed implicitly, and that the CDIM method possibly would support the process. He remarked that it differs from organization to organization whether the CDIM method is perceived as restrictive: "*in my previous job, I might have wanted to focus on inconsistency discovery*". The individual scores and comments are depicted in Table 8.5.

Perceived usefulness

As stated in the previous section, the expert was not sure of the usefulness of concerns to discover inconsistency in the CDIM method. The matrix was referred to as an effective means, however the participant indicated that the matrix was effective if the right concerns could be collected (see statement 12 in Table 8.5). Next to concerns and the matrix, the usefulness of the CDIM

method is dependent on the usefulness of the individual components. We asked for the perceived usefulness of the situational factors, the diagnosis activity and the handling activity, as those are the most important parts of the method next to the matrix and the concerns.

5. **Finding: the classification questions should be made organization-specific.** The participant indicated that the diagnosis activity could be very meaningful to decide upon an inconsistency, if the questions were slightly more specific for the particular situation. "*The architect should prepare and adapt the questions. And with specific, I mean company-specific: the organization should make them really company specific.*"
6. **Finding: the handling activity is a useful activity of the CDIM method.** The handling activity was received as presenting structure and direction to the possible solving strategies. "*This step helps us to structure the discussion, and make an informed decision with sound arguments. This step presents a solid point of departure for a discussion.*"
7. **Finding: inconsistency monitoring, diagnosis, handling and finalization are useful.** "*Creating insight in inconsistency and handling, - I would like to see that explicit in my organization. I think that is a good way to discuss a problem and think of a solving strategy. I would definitely use this approach.*"

Strengths and weaknesses

The participant had a positive attitude towards the separation of discovery and management. "*If we look at the matrix in the current situation, those concerns can lead to possible inconsistencies: we did not build it yet. These are things that we still have to do, so the focus is on collecting/identifying concerns and discuss possible potential inconsistencies. So clearly, there are two paths within this CDIM method. I think it a strength that the method has been divided into two main parts. We are not really discovering inconsistency, but the we are dealing with and deciding on handling possible inconsistency*". The participant identified several other strengths, which are summarized below.

8. **Finding: the CDIM delivers usable design solutions.** "*The strongest point of the CDIM method is that it is not very theoretical. You have a lot of architecture frameworks that are very theoretical [which is why we not use it]. I think this is very practical and applicable.*" In addition, the participant mentioned that the method yields practical outcomes: "*I think the method delivers practical results, not only a description, but outcomes of a solution. That is a strength of the CDIM method.*"
9. **Finding: the CDIM methods presents insight in the trade-offs that an architect makes.** The participant indicated to have gained more insight in his architecture and especially in the trade-offs that had to be made. "*I think that if an architect fills the matrix in correctly and completely, that the bottlenecks in the architecture - that are often implicitly clear to the architect and his direct colleagues - but not to others - can be explained explicitly. It can be used to explain where and how the inconsistency arises, and what concerns are important. As a means of communication, it is very useful.*"

The expert also identified several weaknesses:

10. **Finding: populating the matrix can feel artificial.** "*Maybe that is because of the fact that we did not had the time to care and explicitly discuss and specify our concerns. But it is something to think of: how easy and fast is it to fill those matrix axes?*"

11. **Finding: the 'cause of an inconsistency' is a difficult notion.** "*An 'inconsistency cause' is not really different from a 'conflict of interest' between different aspects or assumptions. A conflict of interest implies the cause of an inconsistency*". "*Furthermore, the cause is often known when you are searching for conflicts between concerns. It is still good to mention it explicitly.*"
12. **Finding: the steps in the activity *Concern Monitoring* are intertwined and inseparable.** The participant argued that during a workshop/method it is good to have explicit steps. "*In a workshop, having such explicit steps is good: "first I am going to determine wh́at to discuss instead of immediately diving into design alternatives". Then you decide what to focus on prior to solving it. In reality however, I'm doubting if the (two) steps are performed separately*".
13. **Finding: stakeholders' roles need to be made more explicit in the CDIM method.** Who is responsible for what action? A responsibility assignment matrix (RAM) was one of the suggested approaches. In a RAM, the responsibilities of each involved stakeholder in a project or business process is described [157].

8.3 Results of the Expert Interviews (Experts 3 and 4)

The two expert interviews that have been performed yielded various results. The uninterpreted and raw results are described below.

8.3.1 Expert interview 1 (Expert 3)

The in-depth walkthrough of the CDIM method and the illustration of the screens led to series of comments and questions. In general, the expert had a positive attitude towards the prototype. He indicated that he must be able to see a detailed overview of an inconsistency in a cell: "*I want to see priorities of the concerns related to an inconsistency. On the basis of those priorities I decide what should be the handling decision.*" He mentioned he needs a rationale in order to recall why he made a certain decision, or why he assigned a certain value to an inconsistency. The expert stated that he appreciated the comprehensive overview, but that he wants to have the details when he clicked on a cell or concern.

Requirements satisfaction

In general, the expert rated the statements positively. The individual ratings and comments are depicted in Table 8.6. He appreciated the statements with a positive attitude[2]. The expert was moderately doubtful on the steps and phases of the method, as he indicated that he regularly needs some time when he sees a new method or technology for the first time. The participant also remarked that the length of the cycle relies on the experience of the architect, and the size of the matrix, and the available architectural documentation. The expert had enough techniques available to execute the method. With regards to stakeholders, the expert mentioned that diversity of stakeholder types is more important than the number of stakeholders.

[2]The negatively phrased statements were recoded.

	Statement	Rating	Comments (expert interview 1)
	1	4	-
	2	5	"The method is usable during the entire architecture and development process, from contracting to implementing a system." It is not relevant in 1 specific phase in the project, more likely you use it continuously.
	3	4	Each architect has techniques available for prioritization and classification. Otherwise you cannot call yourself an architect.
▼	4	1	Stakeholders are important in each process. It is important to have sufficient variety of involved stakeholders. They are not all equally important, but they all are somehow important during your entire process.
	5	5	We use a tool for that as well: all important conflicts and bugs are stored. If you experience a problem, people search in that tool for earlier similar experiences.
	6	3	Neutral, as I currently have a slightly different maintenance focus.
▼	7	1	Method does not feel compelling, since I will select and pick out the elements that are beneficial for me. If those elements support you as an architect, I am willing to do some extra steps.
	8	4	The CDIM method adds structure to the process.
▼	9	3	I cannot say that explicitly, I haven't used the method.
	10	4	Those loosely coupled elements make the CDIM method flexible.
	11	4	Concerns let the architect focus on what he needs to solve, and what his problems are. If you outline those concerns, you will be able to problems earlier.
	12	4	A matrix is an effective way to do this, if you manage to fill in the axes of the matrix correctly, thus finding the right concerns and using them correctly.

Table 8.6: Responses of expert 3.

1. **Finding: the activities *Concern Prioritization* and *Concern Modeling* should be performed iteratively and cyclic.** During the walkthrough, the expert proposed that after gathering concerns, prioritizing and specifying can be done iteratively, because the prioritization dependent on the the specification of a concern.
2. **Finding: stakeholders should be introduced during categorization.** "*Categorizing concerns should be done with the involvement of important stakeholders. Stakeholders can be found on every aspect of the organization: even the infrastructure-department could be a stakeholder.*"
3. **Finding: the CDIM method can be applied in different phases.** The expert indicated that the CDIM could be used continuously: "*It is not a method you execute at one specific moment in time, but you can use it continuously, as you also run into problems continuously.*"
4. **Finding: the CDIM method adds structure to the process of inconsistency management.** The expert predicted that the CDIM method would add structure to the process of inconsistency management, and would not be restrictive and constraining (see statements 7, 8 and 10).
5. **Finding: the CDIM method does not constrain an architect.** "*The method does not feel compelling. I like to pick out the components that I feel are necessary and useful. A method needs some prerequisites to be satisfied in order to support you. If you think the method is useful and supportive, than you should take those preconditions and guidelines for granted. Just pick out the things you need to achieve your goal. It is a feeling: and the*

CDIM method does not feel restrictive. If I am able to apply this method in my job and it benefits me, than it is fine if I have to perform certain steps or do some pre-work. If I want to drive my car, I need to do certain pre-activities as well?"

Perceived usefulness

The expert described concerns as the building blocks for an architecture, the things that push an architect towards 'what should be built', 'what should be solved', and 'how to approach those problems'. He indicated that if the architect is supported with aligning those concerns structurally and clearly, than the CDIM enables the architect to to detect possible inconsistencies and possible complications.

The expert indicated that he would try the method, but explained that he was dependent on others if he wanted to adopt the method within his daily practices: "*If I am at a client, I am strongly dependent on their way of working, and the method should then be applicable at a client's side [...]. But as an architect, I would say, this method offers extra grip with respect to my activities and tasks. I would definitely like to gain experience with the CDIM method.*" When asked if he would recommend the method to the client, the expert indicated that he could not say an explicit yes or no. He mentioned that proposing a new method for a client is possible, however, the architect that does this has to have a track record and various positive experiences with the method. "*I would certainly involve the CDIM method in my evaluation, but I do not know whether that will result in a recommendation for the company: that is strongly dependent on the situation.*"

He adopted a possible a positive attitude towards the usefulness of the CDIM method:

6. **Finding: concerns are an effective way to detect inconsistency.** "*If you manage to align the problematic concerns correctly, you will be able to use those relations in order to detect possible problems earlier. Without doubt, there will be concerns that are interfering. If you can represent that clearly, it could certainly help you to address those inconsistencies.*"
7. **Finding: a matrix is an effective means to discover overlaps and inconsistency.** Given that the axes of the matrix are filled out correctly, the architect stressed that a matrix is an effective approach to outline different concerns.
8. **Finding: the CDIM profiles create a unified vision among the involved stakeholders.** The two 'method profiles' (pragmatic and systematic) were perceived as useful by the expert, as he indicated that it is important to establish the situation in which a method is executed. "*Usually, these aspects are a given. Regardless, it is important to establish them throughout the involved people, as that creates a unified vision*".

Strengths and weaknesses

Evaluation of the CDIM method resulted in various identified strengths:

9. **Finding: the CDIM method is applicable during all phases of a project.** The expert stated that he would focus more on one part of the method dependent on in which phase the current project is. "*That makes the method implementable, since you can use the method next to existing development methods, and that is quite important.*"

10. **Finding: the CDIM method aligns with many ways of working.** The expert mentioned that he saw "*similarities with agile- and architecture principles*". As a result, he stated that the CDIM method will correspond to and align with many ways of working.

The expert also identified a key weakness:

11. **Finding: tool adoption and tool understanding are limitations of the CDIM method.** "*Even if you solely use a spreadsheet, how do you get people to use it?*" He argued that adoption and unrolling a method within an organization is a difficult aspect, with many different stakeholders, "*and eventually they should be able to use and understand the method*". "*To embed such a method in an organization, that takes time. Often, there are many techniques and ways of working available, so it will take time. And time equals money, so it will not be easy.*"

When asked how the architect would assess possible cognitive overload he responded that possible overload depends on the experience of the architect that is executing the method, next to something such as the screen size, or matrix size. "*You should not select too much concerns, otherwise a practice like this misses its purpose. It is called a shortlist for a reason.*"

8.3.2 Expert interview 2 (Expert 4)

Several comments and advices emerged during the walkthrough of the method. The expert adopted a critical view towards the concerns and mentioned that the CDIM method should let the architect itself determine the definition of concerns. Additionally, he argued that concerns could be 'specified' as user stories, in order to make them more understandable and concrete for non-technical stakeholders. With respect to the notion of 'perspectives' the architect mentioned that the architect should maintain a sort of standard perspectives that can be reused over time. When discussing the matrix, the expert indicated that a systematic architect would perhaps follow the steps of the CDIM's last three activities, but a pragmatic architect imaginably does the final three steps at once when he finds a conflict or inconsistency.

Requirements satisfaction

The ratings given by this expert are the highest, compared to the two case studies, which indicates a positive attitude. The expert gave little comments on his statements, and even when asked for comments, he occasionally indicated that he had no real valuable comments. The phases and the process support of the CDIM method appeared clear to the expert. The expert indicated he had seen to little depth of the method to be able to assess whether a design rationale and a knowledge base are effective means to promote reusability. The individual scores and comments are presented in Table 8.7.

1. **Finding: the CDIM method should be user-centric.** The expert explained that the involvement of stakeholders in the CDIM method is essential.
2. **Finding: the CDIM method does not restrict the architect's architectural freedom.** The expert was optimistic about the flexibility and the difficulty of the method. "*I see it as an instrument that I choose to use - or not.*"

	Statement	Rating	Comments (expert interview 2)
	1	4	The steps and guidelines are clear, as you told them.
	2	5	-
	3	5	-
▼	4	1	The user should be centric. We always involve stakeholders or users in the process. It is essential to involve them.
	5	4	I feel I do not have enough experience with the CDIM method in order to give a reasonable answer.
	6	4	The method seems tailorable, so it should fit into your daily activities.
▼	7	1	-
	8	5	The CDIM method does not constrain my freedom as an architect, I see it as an instrument that I choose to use - or not.
▼	9	1	-
	10	5	The method seems very flexible.
	11	4	Concerns might be seen as high level requirements that are the building blocks of an architecture. If concerns are seen that way, using business value and impact is a good way of classifying: which is an added value. One might also do this with conflicting requirements.
	12	5	A matrix is an easy and comprehensive way to outline the concerns.

Table 8.7: Responses of expert 4.

Perceived usefulness

At a first glance, the expert did not perceive concerns a useful means for identifying inconsistency. However, the architect came up with example concerns from a security perspective: preventing a hacker from accessing the system, preventing disclosure of unauthorized information.

> "*If you see concerns as building blocks for the architect, as high level requirements, which eventually lead to design decisions, and you outline the design decisions, than you can indeed use those classification factors, and this might work. And if you view the architecture through a certain 'perspective', you end up with certain concerns that can be translated into requirements and eventually into design decisions. Frankly, an inconsistency between design decisions is what will eventually be problematic for you as an architect.*"

The expert indicated to have no intention to use the situational factors, but he was very positive about the usefulness of the CDIM method when applied to development teams:

3. **Finding: the CDIM method can be used as an instrument to assist teams with feature development.** The architect voiced the idea to use the method as a standard 'checklist'. "*When working on certain features,*" he stated, "*the teams can use this a structured guideline to check new designs and implementations through certain standard perspectives on conflicting concerns and requirements.*"
4. **Finding: inconsistency diagnosis should be tailored towards the needs of the organization.** The architect indicated to start off the classification with asking if the inconsistency will arise again or frequently, in order to assess the severity of the consequences. "*If it happens frequently, that means that the risk of occurrence of such an inconsistency is high right?*"

5. **Finding: classification can be problematic if the cause of an inconsistency is not known.** The expert argued that if the cause and the type of the inconsistency are not known, it can be difficult to assess the other aspects business value, impact and engineering effort.
6. **Finding: the CDIM method is usable on the level of user-stories.** "*I think it is feasible to devise a set of prefab perspectives and then think of requirements or concerns that can be used in a matrix. On the level of user stories, this method could be useful.*"
7. **Finding: method adoption can be a limitation of the CDIM.** "*If you want to do CDIM within an organization, all architects or an architecture board should initiate the method, and concerns should be much more high level, at least not too low-level: otherwise you will drown in the amount of concerns.*"

The expert expressed an intention to use the method, albeit a simplistic version of the CDIM method that can be used by self-managing teams. Those teams have a definition of to-do, which helps the teams to identify conflicts, by looking from different perspectives to the (in this case) 'story'. "*At a low-level user story-level I see the advantages relative to how we are currently working. By giving the teams a tool such as the CDIM method they can take charge and do initial evaluation, which could prevent a single trip to the hazard analysis board.*"

Strengths and weaknesses

The expert illuminated a possible strength and a possible weakness of the CDIM method:

8. **Finding: the CDIM method enables earlier identification of inconsistency.** "*I think if done correctly the method can help you with structuring the development of new features. The point is right now, we discover inconsistency on the go: we discover inconsistency during design, implementation and after implementation. If you find them earlier, this could certainly prevent some rework. In my opinion, that is an advantage of the method.*"
9. **Finding: the CDIM method must be tailored in order to be used.** "*Tailoring is absolutely required: defining concerns, perspectives, actions etc.*" The architect argued that if that does not happen, there will be discussions about what is a concern and what not. "*Getting the definitions straight should be a meta activity before you start the method. Especially because tailoring is so important, you should explicitly let the architect do that.*"

8.4 Results Compared

Both participants in the case studies as well as participants in the expert interviews were presented the same statements and questions, in order to see whether participants expressed a different attitude in the two types of evaluation. The number of participants is four, which logically prevents the use of any statistical approaches. The results have been summarized in Table 8.8. The evaluation provided no reliable new insight into differences in attitude, as the scores per statement do not differ much. The scores depicted in the table are Likert scores and, as such, an average score may not be calculated since Likert scales represent ordinal data. Table 8.8 shows that statement 1, 5, 7, 9, 10, 11 led to 'more positive' answers among interview experts. Only one statement led to a higher appreciation by case experts: statement 6 (whether the

		Statement	Case 1	Case 2	Interv. 1	Interv. 2
	1	The guidelines and process support is clear.	3	4	4	4
	2	It is clear to me in which stage the method should be applied.	5	5	5	5
	3	I have a technique for each step if it is required.	5	4	4	5
▼	4	It is not feasible to identify and involve stakeholders in this process.	1	1	1	1
	5	A design rationale and a knowledge base are effective means to promote reusability in this process.	3	3	5	4
	6	The method fits in my daily practices as a software architect.	4	4	3	4
▼	7	The method feels restrictive and compelling.	2	1	1	1
	8	The method does not constrain my architectural freedom.	5	4	4	5
▼	9	The method is difficult.	3	2	3	1
	10	The method is flexible.	4	4	4	5
	11	Concerns are an effective way to detect inconsistency.	4	3	4	4
	11	Using a matrix is an effective way of detecting overlaps.	4	5	4	5
		(1=strongly disagree, 5=strongly agree, 3=neutral)				

Table 8.8: The responses of the case experts and the interview experts compared.

method fits in the daily practices of the architect). The rest of the statements led to an 'equal' appreciation. With high prudence, we carefully infer that interview experts adopted a more positive attitude towards the CDIM method. If this is truly the case (which cannot be verified from the data), this might be explained by the fact that the interviews have not really worked with the CDIM method, and have thus not experienced the possible downsides. As we are cautious with drawing conclusions from this limited data, we do not further analyze the comparison between the statements anymore.

8.5 Suggested improvements

Several improvements have been suggested resulting from the evaluation.

- In the *Planning activity*: present two main goals, but present the freedom to let the architect choose its own goal (Exp.1).
- Introduce explicit roles for each activity (Exp.2).
- Include a meta-activity in the method, to enable the architect to tailor it (Exp.2,4).
- In the *Concern Prioritization* activity, introduce a specific guideline on the length of the concern shortlist (Exp.2,3).
- With respect to the matrix: each cell should contain the candidate solutions, if the inconsistency is flagged as solved (Exp.1)

- An initial fill of the matrix as an example should be attached to the CDIM method (Exp.1,2).
- The step *Monitor inconsistency* should indicate explicitly how a decision should be monitored (Exp.1,4).
- Include a guidelines for explaining the method to other architects (Exp.2).
- Include explicit feedback loops (Exp.1).

Together with improvements resulting from the requirements satisfaction, these suggestions will be discussed in Section 9.6.

9 Synthesized Findings

This section synthesizes and discusses the results of the evaluation, guided by the *requirements satisfaction*, *the strengths and weaknesses*, the *perceived usefulness*, and the *intention to use* the CDIM method.

9.1 Requirements satisfaction

Since the requirements formed the starting point for the development of the CDIM method, the degree to which the requirements have been addressed indicates the success of the development efforts. The set of 23 requirements contains 11 requirements originating from expert interviews and 12 requirements originating from literature. A subset of the requirements is assessed by the statements presented in Table 8.2. The other requirements are assessed during development of the CDIM method. We briefly discuss each requirement and present arguments to support its degree of satisfaction.

9.1.1 Requirements from the expert interviews

Requirement ER1: Satisfied. Statement 9 demonstrates that the CDIM method is perceived as simple and consistent in use, however it was perceived to have certain shortcomings in terms of clarity, which can be explained by the fact that 2 experts did not use the method but only analyzed it during a walkthrough.

Requirement ER2: Satisfied. The method is generic, but the situational factors and the definition of a shared vocabulary enable the architect to adapt the method to his specific needs.

Requirement ER3: Partially satisfied. Architects indicated that tool support should not be with a separate tool. To address this, a spreadsheet template was constructed that can be used in Microsoft Excel. Besides that, a conceptual prototype tool has been designed to support architects during the method. Support for an architectural tool is absent however.

Requirement ER4: Partially satisfied. The evaluation showed mixed results: experts had an intention to use (parts of) the CDIM method, saw an increased insight in the architecture and communication on one side, but identified certain obstacles as well.

Requirement ER5: Satisfied. Woods and Hilliard [156] indicate that an important reason for adoption failure is a mismatch between the ADL and the architect's way of working. The CDIM does not force the architect to document everything. Results on statement 7 and 8 indicate that the method is not restrictive or compelling, and that the architect is not constrained in his architectural freedom. The CDIM method was perceived as very supportive for explicating and structuring the process of inconsistency management.

Requirement ER6: Satisfied. The experts indicated that the CDIM method is flexible, mainly due to the loosely coupled iterative activities. It enabled the experts to execute certain parts of the method and avoid other parts. Two experts mentioned the term 'toolkit approach', which is one of the architecture principles that formed input for the method (AP5 in Sect. 5.3).

Requirement ER7: Satisfied. The fact that the method is cyclic and needs to be iteratively executed, invites the architect to incrementally improve and develop the matrix. Experts argued that the CDIM method can be executed at every given moment.

Requirement ER8: Partially satisfied. Cognitive overload is prevented as much as possible by categorizing the concerns in perspectives, using a matrix, and advising on the length of the shortlist (limiting the resources) [86]. However, cognitive overload depends also on the size of the matrix, the number of concerns selected, and in some cases the screen size of the architect's device.

Requirement ER9: Satisfied. Hierarchy is introduced by grouping concerns in perspectives, and in the conceptual prototype by hiding details under 'hover actions'.

Requirement ER10: Satisfied. The architect is able to analyze a set of concerns iteratively when he needs it. The matrix does not form a large pile with outdated information. Prior to a discussion, architects can construct the matrix again with new, up-to-date information.

Requirement ER11: Satisfied. This is up to the architect that executes the CDIM method: a spreadsheet is accessible by everyone with access to the internet if it shared.

9.1.2 Requirements from literature

Requirements from literature on architectural evaluation methods:

Requirement AE1: Satisfied. Especially important are guidelines on when actions need to be executed and in which sequence [58]. Statement 1 indicates that the guidelines and process support are clear. Explicit guidelines on the role of involved stakeholders were indicated as missing, but these are unnecessary as stakeholders are only involved in discussions.

Requirement AE2: Partially satisfied. Statements 2 and 3 indicate that experts understood when the CDIM method could be applied and the CDIM method presents clear techniques and approaches. The level of information that is required as an input is an important aspect [5], but was not clear in all cases.

Requirement AE3: Satisfied. Experts agreed that active stakeholder participation is essential for high-quality output [26], but also indicated that it is not always feasible. It is necessary for a method to identify key stakeholders and their objectives [5], which is a step in the CDIM method.

Requirement AE4: Satisfied. Kazman stated that the importance of addressing non-technical issues is becoming more apparent [82]. The situational factors address this, by urging

the architect to adapt the method to non-technical issues like organizational structure or communication.

Requirement AE5: Partially satisfied. The CDIM method is validated in lightweight case studies and expert interviews, which have their limitations (see Sect. 10.1). Studying the CDIM method in other contexts and on longer terms might result in more validation [5, 39].

Requirement AE6: Partially satisfied. Reuse is recognized as a critical means of gaining quality results [3] and should include document templates, design rationale, etc. [150]. Statement 5 indicated that the knowledge base was perceived better than the design rationale as a means for reuse.

Requirements from literature on architectural description languages:

Requirement AL1: Partially satisfied. Statement 11 indicated that concerns are difficult to grasp and specify. The evaluation also demonstrated that some concerns exist implicitly in the minds of the architect, in line with research of [93]. The CDIM method supports the modeling of specific concerns, but concerns have their limitations.

Requirement AL2: Satisfied. The method supports multiple views by using perspectives to categorize concerns in certain sets. This enables architects to use multi-view analyzes, which is seen as a necessity in ADL literature [103].

Requirement AL3: Satisfied. Statement 6 illustrates that the CDIM method fits in the architects' daily activities. Experts mentioned that the complete method may not be applicable in all situations, but especially the final four steps were perceived as suiting the daily activities. Experts expressed that the iterative and modular setup of the CDIM method contributes to these findings.

Requirement AL4: Satisfied. This requirement has overlap with ER5. The architect is not forced to document everything. Results on statement 7 and 8 indicate that the method is not restrictive or compelling, and that the architect is not constrained in his architectural freedom.

Requirement AL5: Partially satisfied. There is overlap with ER3. Authors [154] show that tool support is important as integration to existing tools, which we aimed to satisfy by making an Excel template.

Requirement AL6: Satisfied. The CDIM method's modular and iterative design enables architects to incrementally adopt the method, without being forced to switch to the CDIM method at once in full. The chances of success in adoption are higher if architects can apply it incrementally to existing work [154].

In total, 15 of the 23 identified requirements are addressed satisfactory. This suggests that development of the CDIM was accomplished; the CDIM method aligns with best practices from the field of architectural evaluation; the CDIM can be implemented in architects' practices; and the CDIM needs some improvements. A set of 8 requirements have not been completely satisfied. This can be explained by the fact that the satisfaction of these requirements is somewhat difficult to assess in some cases, in which we decided to mark the requirement as 'partially satisfied'. The 8 requirements present us with concrete areas of improvement. These areas are: improvement of tool support and integration (ER3 and AL5), better definition of the method's input (AE2 and AL1), improving cognitive overload (ER8), increase reuse (AE6) and performing long-term validation (ER4 and AE5).

9.2 Strengths of the CDIM method

The evaluation shows that the CDIM method has various strengths. Synthesized findings are supported by findings from the evaluation, in which case the number and expert are annotated (e.g. Exp.1 for results of expert 1). The strengths are summarized as follows:

The CDIM method yields usable design solutions. Findings 7 and 8 (Exp.2) and finding 8 (Exp.4) indicate that a major strength resulting from the evaluation is that the CDIM method yields practical results in the form design solutions, instead of pure descriptive results such as where and how many inconsistencies have been detected. This enables the CDIM method to be used for trade-off analysis, and to be used as a design rationale for design decisions related conflicting concerns. Tang [141] illustrated that software architects often do not document design rationale as they have no time or budget, no standards, or no suitable tool at their disposal [141]. Architects struggling with documenting a design rationale can benefit from this when using the CDIM method. Although this is a side-effect of putting the method in practice, it is a significant outcome of the evaluation.

The CDIM method adds structure and explicitness to the inconsistency management process. Findings 2, 3, and 6 (Exp.1), findings 1 and 6 (Exp.2) and finding 4 (Exp.3) demonstrate that the CDIM method adds a lot of helpful structure to the process. A lot of assumptions and steps that architects would usually have or perform implicitly, were explicated by the CDIM method. Experts argued that they appreciated that the CDIM method 'really structures' the process and forces the architects to deal with inconsistency management consciously. This is a strength of the method, as ad-hoc inconsistency management can be erroneous. An expert mentioned that if correctly executed, the method helps identifying conflicts and inconsistencies earlier and can thus prevent rework.

The CDIM is useful for trade-off analysis. The CDIM method may not be optimal for detecting tangible inconsistencies (in documentation or models), it appeared to be that the CDIM method is especially useful for identifying trade-offs and other conflicts through the use of concerns and the matrix. This is supported by finding 9 (Exp.2) and finding 3 (Exp.1). During the case studies, it was observed that the participants came to new insights with regards to the architecture. This might indicate an increase in the architect's productivity, which contributes to the usefulness of the CDIM method. Our observations are backed up by the participants, that indicated to came to design solutions and ideas. However, since the lightweight case studies have limitations, it is difficult to infer whether the participants' insight had really increased. Appreciation of statement 12 indicates that the matrix is perceived as a comprehensive way to detect overlapping and conflicting concerns, given that the right concerns are selected. It could be improved by providing a template with the CDIM method that shows how a matrix can be populated and used.

The deliverables of the CDIM method can be used as a design rationale. The case studies demonstrated that the CDIM method is quite useful as a communication instrument. Finding 4 (Exp.1), finding 1 (Exp.3) and finding 9 (Exp.2) support this. This makes us think that

the matrix, resulting from the CDIM method, forms a *design rationale* in itself. This is supported by the results of the case studies, which both yielded design solutions. The CDIM method promoted discussions among architects, thereby embracing some of the positive implications of inconsistency. Positive consequences of inconsistency are: reflecting conflicts between different perceptions of stakeholders involved; indicating architectural aspects that deserve more attention; or promote inspection of design alternatives [139].

The CDIM has a flexible character. Finding 1 (Exp.1), finding 1 (Exp.2), and findings 3, 5, 9 and 10 (Exp.3) indicate another strength of the CDIM method. That is that its modularity enables the architects to use the CDIM method with a 'toolkit approach' [129], referring to the fact that architects use those elements of the toolkit they need and ignore the rest. The lightweight and flexible character of the CDIM method was perceived positively during the evaluation. In combination with the applicability during all phases of a project, the modularity makes the CDIM method a practical instrument.

The CDIM's situational factors create a unified vision among involved stakeholders. Finding 8 (Exp.3) shows that the feasibility of performing the CDIM is determined by the context in which such a method takes place. This is in correspondence with findings of Bass and Nord [9]. Experts indicated that the situational factors, which are the basis for tailoring the CDIM method to its context, are useful in order to set everyone involved off in the right direction. Although the factors are often *given*, it is wise to discuss them among the involved people in order to create a unified vision. The quality of the individual situational factors could not be validated.

A human-based collaborative method is the only method for detecting inconsistencies in informal models. A concern-driven method logically asks for stakeholder involvement. Finding 2 (Exp.2), finding 2 (Exp.3) and finding 1 (Exp.4) showed that involvement of relevant stakeholders is desired, but not always feasible. Involvement of stakeholders for analyzing an architecture introduces extra 'weight' to a method [5]. How many stakeholders are involved during the CDIM cycle should be in balance with the time available. Stakeholder-centric methods for inconsistency management involve human inspection of overlap between elements, and human-based collaborative exploration [139]. Approaches such as Synoptic [41] and DealScribe [127] are related to the CDIM method, but differ from it as they cannot be used for other types of (implicit) inconsistency. Synoptic requires stakeholders to specify conflicts in so-called 'conflict forms' to describe conflicts that exist in models, which have minor similarity with the 'concern-cards' used to specify concerns in the CDIM method. In DealScribe, stakeholders look for 'root-requirements' in their models. Root requirements are identified for concepts present in the models, and pairwise analysis of possible interactions between root requirements results in a list of conflicting requirements. A limitation is that pairwise sequential comparison is time-consuming and labour-intensive. We have aimed to mitigate this limitation by forcing the architect to set a scope on the architecture under analysis. Despite being time-consuming, stakeholder-centric and human-based approaches are the only method for detecting inconsistency in informal models, and models expressed in different languages [139].

The CDIM can be used complementary to architecture evaluation methods. Furthermore, even though architecture evaluation methods such as the Scenario-based Architecture

Analysis Method (SAAM) [84] and the Architecture Trade-off Analysis Method (ATAM) [85] have a different direct goal than the CDIM method, they are related (see Sect. 4.10). As the CDIM method can also be applied *during* the architecting process, and SAAM and ATAM require the architectural design to be present [5], the CDIM method provides alleviation of the resources required for performing the ATAM and SAAM. The reason for this is that conflicts may possibly already be discovered or prevented in an earlier phase. In this sense, the CDIM method may be used complementary to an organization's existing architecture evaluation method(s).

9.3 Weaknesses of the CDIM method

The CDIM method requires tailoring. Findings 3, 7 and 8 (Exp.1), findings 4 and 5 (Exp.2) and findings 4 and 9 (Exp.4) indicate that the CDIM method is not flawless. If the CDIM method is not tailored, it is too generic and imprecise to use right away. If concerns are defined on the wrong level, the CDIM might not be applicable in some situations. This is mitigated by adding an activity to the CDIM method that enables the architect to creates his own vocabulary and definitions However, the architect must tailor several activities in order to use the method. Embedding the method in an organizational context requires time and active involvement of an architect, which generally costs time and resources. In addition, involved (non-technical) stakeholders need to understand the selected concerns and the tooling. Even though this is addressed by several design choices concerning the CDIM method, this may create potential barriers for adoption.

The effectiveness of concerns for detecting inconsistency is debatable. Finding 7 (Exp.1) and findings 3, 9 and 10 (Exp.2) show that the effectiveness of concerns for detecting inconsistency is debatable. Even though several authors underline the importance of concern-centric architecting [67], architecting in the face of multiple stakeholder concerns [95], and viewing concerns as primary drivers of software architecture [129], the effectiveness of using concerns for detecting tangible inconsistency is debatable. Nonetheless, concerns appeared to be useful for identifying overlaps and conflicts, but only when the concerns are collected on the right level of detail. Both case studies demonstrated that the definition of concerns led to confusion among stakeholders and architects. If the concerns and perspectives are defined on the wrong level, the method might not be applicable or produce inaccurate results. If concerns are too high level, it might be difficult to associate design decisions with them, while those connections between design decisions and concerns appeared to be very critical for finding inconsistencies or conflicts.

Populating the matrix feels 'artificial'. Findings 3 and 8 (Exp.1) and finding 10 (Exp.2) showed that populating the matrix with concerns can feel a little bit 'artificial', which is a considerable weakness of the CDIM method. Experts indicated that they often knew which kind of inconsistency or conflict could arise, however they found it difficult to capture such a conflict in terms of concerns. To address this, three experts proposed to include the 'tailoring-step' in the CDIM method, in which the architect devises its own definition for the notion of 'concern' and clearly communicates this with involved stakeholders. Furthermore, the matrix is two-dimensional, indicated as a limitation in the sense that an architect can only analyze two perspectives simultaneously.

9.4 Perceived usefulness

Experts adopted a positive attitude towards the usefulness of the CDIM method (e.g. findings 3 and 7 (Exp.2), or findings 3 and 6 (Exp.4). The CDIM method is found to be a promising approach since it helps the architect to structure and explicate the inconsistency management process, and could also act as a guideline in discussions on concerns during the architecting process itself. The CDIM method especially aims to be useful in contexts where little focus and time is available [155] as it is light and does not require extensive effort to be done prior to detection of overlaps. This is opposed to formal approaches and techniques, which require formalization of some kind [138] or construction of rule sets [111]. Not all activities of the CDIM method were perceived as useful. The outcomes illustrate that *Inconsistency Diagnosis* contributed to structuring and explicating the decision making process. However, the constructs used to classify an inconsistency must be tailored to the specific context of the organization in question. The scores attached to each question were found arbitrary and not useful: experts indicated that high, medium and low will suffice. The decision model in *Inconsistency Handling* proved to be fairly effective because it provides a well-analyzed rationale for deciding on an inconsistency. The handling step helped the architects making a sound, informed decision and provides a solid point of departure for discussion. Overall, the experts found the CDIM method primarily useful for detecting conflicts and inconsistent design decisions, and less for detecting tangible inconsistencies. As a result, we imply that the CDIM method could be best performed complementary to other approaches that focus on for instance more on model inconsistency.

9.5 Intention to use

Participants of the case studies indicated to have an intention to use the CDIM method. The first case expert was enthusiastic, while the second case expert indicated he would use especially the matrix and the last three activities of the CDIM method. We noticed that experts preferred usage of the matrix, while being less enthusiastic on building the matrix, which happens in the first few steps of the CDIM method. This can be explained by the fact that it takes some effort to construct the matrix, while most experts were very pragmatic and result-oriented. An explanation for the fact that they were enthusiastic about using the matrix is that architects indicated they would like to see more formality in their organization's inconsistency management process. One expert argued that he saw considerable benefits in using the CDIM method for structuring individual teams, who may use the perspectives and the matrix to detect inconsistent decisions while developing new features.

9.6 Towards the final CDIM method

The evaluation resulted in several areas of improvement (Sect. 8.5). The analysis of the requirements satisfaction also led to 5 areas of improvement (Sect. 9.1). This section presents changes to the evaluated version of the CDIM method, that lead to the final version of the CDIM method (depicted in Fig. F.10 in Appendix F.2).

1. Present two main CDIM goals, but enable architects to choose their own (Exp.1).

2. *Prioritization* and *Modeling* are highly intertwined and in practice inseparable (Exp.3).
3. Introduce explicit roles for each activity (Exp.2).
4. Include a meta-activity in the method, to enable the architect to tailor it (Exp.2,4).
5. Introduce a specific guideline on the length of the concern shortlist (Exp.2,3).
6. Indicate explicitly how an inconsistency should be monitored (Exp.1,2,4).
7. Each cell in the matrix should contain the candidate solutions or problems (Exp.1)
8. The CDIM method should contain an initial fill of the matrix (Exp.1,2).
9. Include a guidelines for explaining the method to other architects (Exp.2).
10. Include explicit feedback loops (Exp.1).

From the requirements:

11. Improve tool support and tool integration (requirement AL5 and ER3).
12. Future validation (requirement AE5).
13. Decrease cognitive overload (requirement ER8).
14. Improve clarity of method input (requirement AL1 and AE2).
15. Promote reuse within the CDIM method (requirement AE6).

9.6.1 Changes

Based on this list, the CDIM method has undergone several changes:

- **Set goal.** The goal setting activity of the CDIM method is now more free (suggestion 1), describing two situations but let the architect choose both goals.
- **Merger.** Concern Prioritization and Concern Modeling have been merged to the activity Construct (suggestion 2).
- **Roles.** Involved stakeholders are now explicitly presented in the CDIM method's practices (suggestion 3).
- **Tailoring.** The method now contains a new activity 'Decide on definitions' that enables the architect to tailor the method towards his specific needs (suggestion 4 and 14). We expect that this improves clarity of the input.
- **Shortlist.** The length of the shortlist is now advised to be 5, as this also decreases the cognitive overload for architects (suggestion 5 and 13).
- **Workshop.** The activity Inconsistency Monitoring has been separated in two activities Identify and Discover. *Identify* is now an explicit workshop in which possible conflicts are identified (suggestion 6). The participants are the architect and relevant stakeholders. Input to this activity is the matrix with populated axes containing concerns. Stakeholders' role is to aid the evaluator with deciding on how conflicts affect the architecture or possibly could affect the architecture, and which conflicts could be problematic. This should always be done in line with the project goal. The workshop consists of the following steps:
  1. Explain the goal of the workshop;
  2. Explain the matrix, its axes and cells;
  3. For each cell:
     a identify importance;
     b identify overlap and possible conflicts;
  4. Summarize results;

The main outcome of this activity is a filled in matrix, presenting the architect with conflicting prioritized concerns. These indicate places where inconsistencies may arise. *Discover* is now an activity to *use* the matrix to find possible inconsistent specifications on the basis of the identified conflicts. We expect that this presents more clarity towards architects (suggestion 9) and it is now more explicit how inconsistency is detected (suggestion 6).

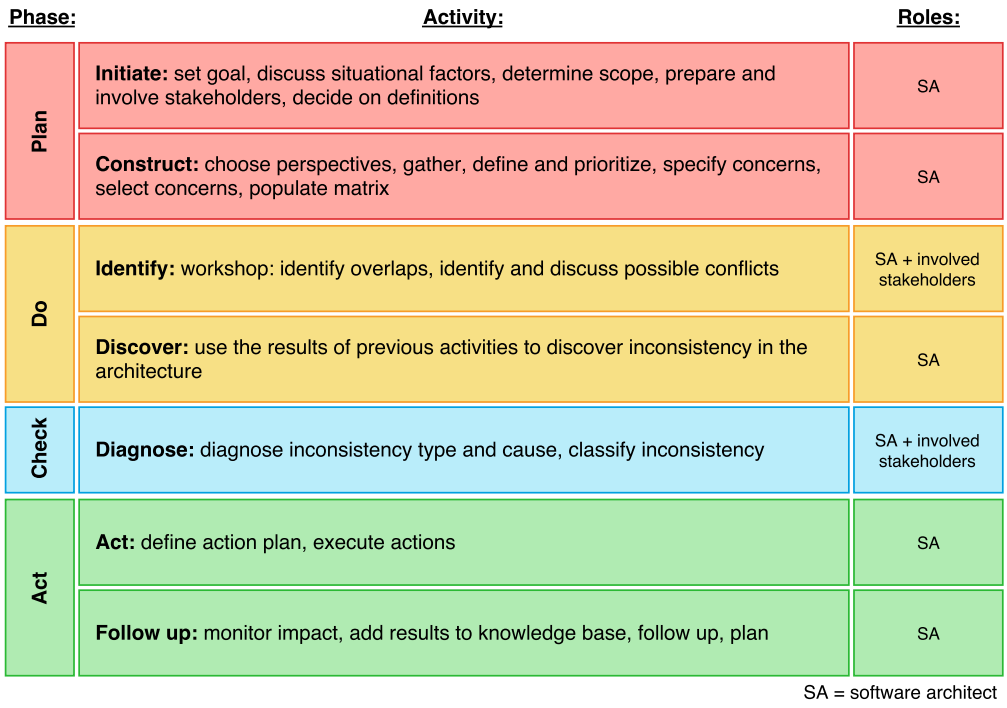


Figure 9.1: The new shape of the CDIM method. As you see in Fig. 9.1 roles have been explicitly presented for each activity (suggestion 3).

- **Name change.** The former 7 activities Planning, Concern Prioritization, Concern Modeling, Inconsistency Monitoring, Inconsistency Diagnosis, Inconsistency Handling and Finalization have been changed to more appealing names: Initiate, Construct, Identify, Discover, Diagnose, Act, and Follow up (suggestion 9, and Fig. 9.1).
- **Phase changes.** This resulted in the three phases Using, Building and Solving to change - in order to closer resemble the activities and the PDCA cycle [79] - to Plan, Do, Check, Act. This also addresses suggestion 10, as the CDIM is now more explicitly cyclic and thus contains feedback loops (Fig. 9.2). The activity Act also contains an explicit step that motivates the architect to reuse certain parts (suggestion 15).

- **Matrix cells.** The cells in the matrix now contain more detailed information in the conceptual prototype (suggestion 7). Several other changes have been introduced. These have been presented in Appendix C (suggestion 11).

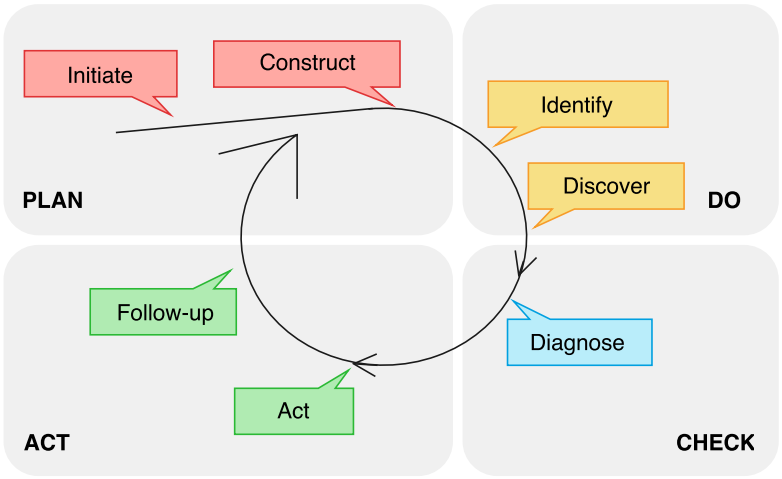


Figure 9.2: The final version of the CDIM cycle.

- **Initial fill.** It was not decided to fill the matrix with examples, but rather present clear guidelines of what concerns and perspectives entail: "The CDIM method defines a concern as: "*a concern about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture*" [130]. However, dependent on the project, the architect should clearly **define and communicate** what a concern exactly means within the context of his work. Furthermore, the architect should define the term 'perspective', as this may differ from organization to organization. It can be kept very broad, however it should enable the architect to categorize the concerns in logical groups, and create a hierarchy in the set of concerns." This decision - together with the activity that enables the architect to tailor the method - lets the architect itself decide upon the definition of a concern in his context (suggestion 8 and 14).
- Chapter 10 and Section 11.1 discuss directions for further evaluation of the CDIM method (suggestion 12).

These changes result in the new, final version of the CDIM method. The complete CDIM method (all steps and deliverables) is represented in the Fig. F.10 in Appendix F.2. Chapter 11 concludes this research and suggests possible topics for future work.

10 Discussion

While this study gained valuable insights, the research and its deliverables are subject to certain limitations. We think that the explication of these limitations is a strength of the research, to the extent that it provides new future research directions and improvements. These limitations can form threats to validity, of which we will discuss three criteria: internal validity, external validity and reliability [158].

10.1 Limitations

Internal validity. Internal validity refers to the establishment of causal relationships throughout the research. In the context of the research, using the same experts for evaluation as for the development of the CDIM method is a threat to internal validity. As experts participated the second time during the evaluation, they were already knowledgeable about the CDIM method, which might have influenced the evaluation outcomes. This is defined as *maturation* or *testing effect*: the fact that the participants become better or wiser during the study's progress [20]. However, we decided to invite the same experts on purpose: introducing and explaining the method to new experts would have required too much time during evaluation sessions. Another threat to the internal validity is that the experts could have behaved differently due to peer pressure resulting from knowing they were part of the research. Besides that, software architects' time and resources are scarce. Finding potential participants was a difficult task: as we invited experts for participation, it became evident that our initial selection criterion of >10 years of experience as a software architect had to be limited to >5 years. If we had managed to invite experts with more experience, this would have likely had a different result. As the time with architects was limited, the CDIM method could not be executed in total. As a result we had to perform 'lightweight' case studies since full executions of the CDIM method would have required more than the two hours we now spent with the architect. To save time and be consistent throughout the evaluation, we created standard forms on which we could write down standard answers and answers on the statements. Moreover, transcripts were processed within 24 hours after the sessions [149], and participating experts reviewed the published transcripts to confirm that we extracted the correct information from the study.

In the context of the CDIM method, internal validity means that the results can be explained

by using the CDIM method and not by other means. The outcomes of the method are highly dependent on the expertise of the architect, the the knowledge and willingness of involved stakeholders, and the documentation available. Though this is clearly a threat to internal validity, these influencing factors are addressed by the situational factors in the first activity of the method.

External validity. External validity refers to the extent to which the outcomes of this research can be generalized to other contexts. One of the many consequences of the limited times and resources of architects is that it was difficult to involve them in the evaluation. As a result, the method has been evaluated in two lightweight case studies and two in-depth interviews. It may be possible that another validation at another organization yields different results in both perceived usefulness as well as the intention to use the CDIM method. The hindered accessibility of architects had another consequence: only one of two 'situations' of the CDIM method has been tested during the case studies. The situation in which the architect aims to detect tangible inconsistencies has not been tested thoroughly. This limitation was addressed by focusing on this open issue during the expert interviews.

Reliability. A study is found to be reliable if it is demonstrated that the results would likely be similar when the study is replicated in a similar context with the same subjects. We aimed to perform a highly reliable research project, by extensively and elaborately documenting the process of the CDIM's construction. For instance, the CDIM requirements from literature have been analyzed on correspondence with the requirements that originated from the experts. Moreover, the features which form input for the MAA, were evaluated by the experts during the first round of interviews. The MAA is a validated approach to construct a new method on existing methods. We purposely only selected candidate methods that have been evaluated extensively in practice. The case studies and the walkthroughs were based on a documented protocol. A limitation with regards to reliability is that the results of evaluation are strongly dependent on the experts' opinions and experiences, which likely evolve over time, threating the reliability of this research.

11 Conclusions

This thesis presents the extensive research on the development and evaluation of the Concern-Driven Inconsistency Management (CDIM) method. Guided by the following main research question, we have spent over 9 months to present an answer and validate this thesis' deliverable:

"*How can a software architect be supported with inconsistency management in software architecture in a structured way?*" Of course, a research question with such a scope requires various sub questions to be answered prior to answering the main research question. We will discuss those questions briefly and if necessary refer to relevant sections in this document.

SQ1: "*What identifies software architecture and the role of the software architect?*"

The term software architecture refers to the set of critical and early design decisions about a system. Stakeholders' interests and concerns form the primary drivers and building blocks of a software architecture. Design decisions address those concerns. There is a spectrum of design decisions: high-level design decisions require informal notations while low-level decisions can benefit from formal notations. As principal and early design decisions are usually high-level, informal models are common in the domain of software architecture. The process of architecting is a life cycle practice that connects problem space and solution space. The role of the architect is ambiguous. It is context- and project-dependent, changes over time, but minimally stretches the definition of the architecture, maintenance of the conceptual integrity, deciding on ASRs and ADDs, assessment and mitigation of risks, participating in planning and development, consulting, informing and collaborating with other stakeholders, and participating in development of a system's vision. Chapter 3 answers this sub research question in full.

SQ2: "*What identifies inconsistency and inconsistency management in software architecture?*"

Inconsistency is present if two or more assertions are made that are conflicting, not jointly satisfiable, or mutually incompatible. Inconsistency arises due to the fact that software architecture is concerned with multiple models, multiple phases, and work is often coordinated among multiple actors; each with their own perspectives, views and opinions. There are many dimensions to inconsistency, of which we distinguish between two main dimensions: tangible and intangible inconsistency. Tangible inconsistency refers to inconsistency in models, specifications, code, and documentation. Intangible inconsistency refers to inconsistent assumptions, concerns, or design

decisions. Inconsistency arises in various forms during the architecting process: in the problem space, inconsistency arises in for instance requirements, models and user stories; during the architecting process, inconsistency in design decisions, assumptions and models is critical; in the solution space, inconsistency in code and models deserves the focus. Architects have to deal with inconsistency in informal models, as these are often used for capturing high-level design decisions. Among tangible forms of inconsistency, various types exist: inter- and intra-model inconsistency, horizontal and vertical inconsistency and inter-phase and intra-phase inconsistency. Inconsistency management practically consists of detecting overlaps, detecting inconsistency, diagnosing inconsistency, and handling inconsistency. Overlap detection is critical, as overlap is a precondition for inconsistency.

SQ3: "*Which means and techniques are available to detect inconsistency in an architecture?*"

Since there are various forms of inconsistency, means and techniques for detecting inconsistency vary greatly. There exist numerous techniques for detecting inconsistency in tangible forms such as models, diagrams or specifications. For intangible inconsistency, the amount of related work is sparse. We discussed several related techniques in Section 4.7, Section 4.9 and in Section 9.2. Furthermore, Table 4.1 presents a comprehensive overview of inconsistency detection mechanisms. For informal models or intangible forms of inconsistency, human-based collaborative approaches are needed.

SQ4: "*What are the requirements of the CDIM method?*"

The requirements formed the starting point for development of the CDIM method. Requirements were extracted from expert interviews, literature on architectural evaluation methods and architecture description languages. The set of requirements focuses on the characteristics of the CDIM method itself, such as the guidelines and process support, the techniques included, stakeholder involvement, the support for multiple views, and its flexibility and practicality. In addition, certain meta-aspects were specified such as tool support, validation of the method, and incremental adoption. The total list of requirements is presented in Section 5.3.1 and Section 5.3.2. Out of 23 requirements, 15 were completely satisfied. The 8 requirements that were partially satisfied presented us with concrete areas of improvement.

SQ5: "*What are the activities and the deliverables of the CDIM method?*"

The answer to this sub research question yields the 'Concern-Driven Inconsistency Management' (CDIM) method, resulting from the MAA, requirements and expert interviews. CDIM is inspired by the Plan Do Check Act (PDCA) cycle and consists of 7 activities:

- **Initiate**: determine aim of the CDIM cycle, discuss contextual factors, set a scope, involve stakeholders and tailor the method
- **Construct**: gather and specify relevant concerns, choose perspectives, prioritize concerns, obtain design rationale, populate matrix
- **Identify:** workshop: identify overlaps, identify and discuss possible conflicts
- **Discover**: use results of previous activities to discover inconsistency in the architecture
- **Diagnose**: determine inconsistency type and cause, classify inconsistency
- **Act**: define action plan, execute actions

- **Follow up**: monitor impact of actions, adjust actions, add knowledge to knowledge base, follow up, plan

In these 7 activities, the evaluating architect studies the architecture of the system in question with the goal to find inconsistencies or conflicting design decisions. Driven by concerns as an input to the CDIM cycle, the architect constructs a matrix, which aids him and involved stakeholders to identify overlaps and discuss inconsistencies. Inconsistencies that are detected during the cycle can be diagnosed and handled, using the detailed guidelines also present in the CDIM method. The CDIM method's last activity enables reuse and helps the architect with the follow-up. The two profiles that can be followed during the method enable the architect to adopt the method in different contexts.

SQ6: "*To what extent helps the CDIM method software architects in managing inconsistency in the software architecture?*"

The CDIM method helps the software architects when (a) the development requirements are satisfied; (b) when architects perceive the CDIM method as useful; (c) and when architects have an intention to use the CDIM method. Not all requirements have been addressed. In total, 8 of 23 requirements have not been completely satisfied. This can be explained by the fact that the satisfaction of these requirements is somewhat difficult to assess in some cases, in which we decided to mark the requirement as 'partially satisfied'. In general, the requirements suggest that development of the CDIM was accomplished, the CDIM method aligns with best practices from the field of architectural evaluation, can be implemented in architects' practices, but needs some improvements. Experts adopted a positive attitude towards the usefulness of the CDIM method, mainly because the benefits of the CDIM method outweigh the weaknesses discussed in Chapter 10. However, not all activities were perceived equally useful, and the CDIM method is more suitable for detecting inconsistent design decisions than for detecting tangible inconsistency. Experts indicated to have an intention to use (parts of) the CDIM method, but stressed that suggested improvements need to be accomplished in order to really embed the method in their environment. Based on the evaluation and a discussion of the findings, we deduce that the CDIM method potentially helps software architects in managing inconsistency in the architecture to a large extent.

RQ: "*How can a software architect be supported with inconsistency management in software architecture in a structured way?*"

The answer comes in the form of the Concern-Driven Inconsistency Management (CDIM) method, inspired by the PDCA cycle. The CDIM method is developed using the MAA and several requirements from literature and experts. Overall the CDIM method was received positively and experts had intentions to use (parts of) the method. Considerable benefits of the CDIM method are the simplicity, flexibility (toolkit approach) and the applicability of the method, in combination with fast, practical results. Results are in the form of design decisions and design solutions. The CDIM method promotes discussion among stakeholders, thereby embracing some of the positive implications of inconsistency. The CDIM increased structure and explicitness of the usually ad-hoc process of managing inconsistency. In particular the matrix and the guidelines for classifying and deciding were found to be helpful for the architect. The CDIM method especially aims to be useful in contexts where little resources and time are available, as it is light and

does not require extensive effort to be done prior to detection of overlaps. This is opposed to formal approaches and techniques, which require formalization or construction of rule sets. Two beneficial side effects in the form of a tool for communication of design rationale and an analysis instrument for trade-offs were found. The CDIM method is certainly not flawless: the method requires tailoring; concerns can be difficult to specify; and the method should offer an initial fill of the matrix, in order to help architects tailoring the method for their organizations.

> The findings suggest that the CDIM enables a software architect to handle and manage inconsistency in the software architecture. For optimal results, the CDIM method should be performed together with other approaches that focus on lower-level tangible inconsistency. We hope to have contributed our part to science and suspect that software architects are supported by the insights provided in this thesis. We believe that we have given a new refreshing 'human' look to the topic of inconsistency management in software architecture. We feel that this is a welcome input in the contemporary context of software architects, as it eventually are the stakeholders and their concerns that are crucial for this domain.

11.1 Future work

The results of this research can be extended in several directions of future research. First, evaluation of the CDIM method was limited in size and depth, as time and available software architects were scarce. There is, therefore, a definite need for evaluation of the final CDIM method on a larger scale. We suggest researchers to conduct multiple-case designs in order to increase the generalizability of the research [158], when resources such as participants, time and money are available. Multi-case studies focus on different cases in different contexts. We believe that a combination of holistic and embedded designs should be used, in order to assess the effects of the CDIM method between different contexts and within different contexts. The contextual factors of software development and software architecture evaluation as described by Bass [9], Clarke [23], Kornyshova [89], and Samer [132] should be taken into consideration in order to assess in which contexts the CDIM method is more effective. Second, continued efforts could focus on building and evaluating the conceptual prototype in practice. The deliverable side of the process-deliverable diagram (PDD) of the CDIM method can be used as a data model on which the tool can be developed (the PDD is presented in Fig. F.10 in the Appendix). The tool should especially consider the role of the architect, and possibly adapt the interface towards the needs of an architect. Perhaps the *systematic* and *pragmatic* profiles (developed as a part of the CDIM method) can be used as a basis to distinguish between different ways of working.

Third, we believe that future practices should focus on the creation of a framework in which multiple complementary approaches can be used. Nuseibeh [116] has already set the stage by developing a generic framework that guides software engineers towards the management of inconsistency. However, this approach uses a central rule repository and is explicitly focused on model inconsistency. There is a need for a framework that is aimed at the industry, and provides a combination of various approaches such as the CDIM method, in order to improve the current inconsistency management practices. We note that this framework should carefully trade off the return on investment of using many combined approaches (which increases the needed resources) and the consequences of missing harmful inconsistencies.

Fourth, more research on the use and specification of concerns as first class entities in software architecture helps us to establish a greater degree of accuracy on this matter. In this sense, we already provided a basis for specifying concerns in the CDIM method: in a template, recording related elements (e.g. design decisions), or in an 'architectural story', (see Chapter 7). Both need validation and coverage in further research, especially a well-defined ontology together with a simple and comprehensive approach towards specification of concerns could be rewarding.

Lastly, a side-product of this thesis is the Design Science Research Cycle (DSRC). Although we have not followed this framework during this research, if is our firm belief that the DSRC provides a comprehensive framework for future design science research projects. The challenge is now to conduct studies in which the DSRC is used, evaluated, and improved on in practice.

Bibliography

[1] P. Abrahamsson, M. Babar, and P. Kruchten. Agility and Architecture: Can They Coexist? *IEEE Software*, 27(2):16–22, 2010.

[2] C. Adelman, D. Jenkins, and S. Kemmis. Rethinking case study: notes from the second Cambridge Conference. *Cambridge Journal of Education*, 6(3):139–150, 1976.

[3] A. Aurum, R. Jeffery, C. Wohlin, and M. Handzic. *Managing software engineering knowledge*. Springer Science & Business Media., 2013.

[4] M. Babar and I. Gorton. Comparison of scenario-based software architecture evaluation methods. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 600–607, 2004.

[5] M. A. Babar, L. Zhu, and R. Jeffery. A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 309–318, 2004.

[6] F. Bachmann and P. Clements. Documenting Software Architectures: Organization of Documentation Package. Technical Report August, DTIC Document, 2001.

[7] R. Baskerville. What Design Science is Not. *European Journal of Information Systems*, 17(5):441–443, 2008.

[8] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Pearson Education, 2013.

[9] L. Bass and R. L. Nord. Understanding the Context of Architecture Evaluation Methods, 2012.

[10] B. Beckert, T. Hoare, and R. Hähnle. Intelligent systems and formal methods in software engineering. *Intelligent Systems and Formal Methods in Software Engineering*, 21(6):71–81, 2006.

[11] P. Bengtsson and J. Bosch. Scenario-based software architecture reengineering. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*, pages 308–317, 1998.

[12] P. O. Bengtsson, N. Lassing, J. Bosch, and H. Van Vliet. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software*, 69(1-2):129–147, 2004.

[13] X. Blanc, I. Mounier, A. Mougenot, and T. Mens. Detecting model inconsistency through operation-based model construction. In *30th International Conference on Software Engineering {(ICSE} 2008), Leipzig, Germany, May 10-18, 2008*, pages 511–519, 2008.

[14] B. Boehm and A. Egyed. Software requirements negotiation: some lessons learned. In *Forging New Links, Proceedings of the 1998 International Conference on Software Engineering, {ICSE} 98, Kyoto, Japan, April 19-25, 1998.*, pages 503–506, 1998.

[15] J. Bosch, M. Gentleman, C. Hofmeister, and J. Kuusela. Software Architecture: System Design, Development and Maintenance. In *IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture (WICSA3), August 25-30, 2002, Montréal, Québec, Canada*. Springer, 2013.

[16] N. Boucké, D. Weyns, R. Hilliard, T. Holvoet, and A. Helleboogh. Characterizing relations between architectural views. In *Software Architecture, Second European Conference, {ECSA} 2008, Paphos, Cyprus*, pages 66–81, 2008.

[17] P. Bourque and R. E. Fairley. Guide to the Software Engineering Body of Knowledge (SWEBOK 3). *IEEE Software*, 16(6):35–44, 1999.

[18] H. Bowman, J. Derrick, P. Linington, and M. Steen. FDTs for ODP. *Computer Standards & Interfaces*, 17(5–6):457–479, 1995.

[19] D. Bredemeyer. *Architect Competency Framework*. PhD thesis, 2002.

[20] M. Brewer, H. Reis, and C. Judd. Research design and issues of validity. *Handbook of research methods in social and personality psychology*, pages 3–16, 2000.

[21] S. Brinkkemper. Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38(4 SPEC. ISS.):275–280, 1996.

[22] S. Brinkkemper, M. Saeki, and F. Harmsen. Meta-modelling based assembly techniques for situational method engineering. *Information Systems Journal*, 24(3):209–228, 1999.

[23] P. Clarke and R. V. O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012.

[24] P. Clement and Others. *Documenting Software Architectures: Views and Beyond. SEI Series in Software Engineering*. Pearson Education, 2002.

[25] P. Clements, J. Ivers, R. Little, R. Nord, and J. Stafford. Documenting Software Architectures in an Agile World. Technical Report July, 2003.

[26] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures.* MA:Addison-Wesley, 2003.

[27] P. Clements, R. Kazman, and M. Klein. The duties, skills, and knowledge of software architects. In *7th Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, page 20, 2007.

[28] P. C. Clements. A Survey of Architecture Description Languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, number March, pages 16–25. IEEE, 1996.

[29] P. C. Clements, F. Bachmann, D. Garlan, J. Ivers, P. C. Clements, L. Bass, R. Little, R. Nord, and J. Stafford. A Practical Method for Documenting Software Architectures. 2002.

[30] A. Cockburn. *Crystal clear: a human-powered methodology for small teams.* 2004.

[31] L. Cohen, L. Manion, and K. Morrison. *Qualitative research methods in education.* Routledge, London, 2007.

[32] M. Cohn. Effective User Stories. In *4th Conference on Extreme Programming and Agile Methods, Calgary, Canada*, pages 15–18. Addison-Wesley Professional, 2004.

[33] M. a. Cusumano and S. Smith. Beyond the Waterfall: Software Development at Microsoft. In *Competing in the Age of Digital Convergence*, pages 371–411. Harvard Business School Press, Boston, MA, 1995.

[34] E. M. Dashofy and R. N. Taylor. Supporting Stakeholder-driven, Multi-view Software Architecture Modeling, 2007.

[35] F. Davis, R. Bagozzi, and P. Warshaw. User acceptance of computer technology: a comparison of two theoretical models. *Management science*, 35(8):982–1003, 1989.

[36] J. Dawes. Do data characteristics change according to the number of scale points used? An experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, 50(1):61–77, 2008.

[37] R. Dèneckere, C. Hug, J. Onderstal, and S. Brinkkemper. Method Association Approach : Situational construction and evaluation of an implementation method for software products. 2015.

[38] E. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66, New York, 1982. Springer.

[39] L. Dobrica, E. Niemela, and E. Niemelä. A Survey on Software architecture Analysis Methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[40] A. Dreibelbis, E. Hechler, I. Milman, M. Oberhofer, P. van Run, and D. Wolfson. *Enterprise Master Data Management: A SOA Approach to Managing Core Information.* 2008.

[41] S. Easterbrook. Handling conflict between domain descriptions with computer-supported negotiation. *Knowledge Acquisition*, 3(3):255–289, 1991.

[42] A. Egyed. Automating Architectural View Integration in UML. pages 1–18, 1999.

[43] S. G. Eick, T. L. Graves, A. F. Karr, I. C. Society, J. S. Marron, and A. Mockus. Does Code Decay ? Assessing the Evidence from Change Management Data. *Transactions on Software Engineering, IEEE*, 27(1):100–112, 2001.

[44] M. Elaasar and L. Briand. An Overview of UML Consistency Management. Technical report, 2004.

[45] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th {ACM} {SIGSOFT} International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*, pages 186–195, 2001.

[46] D. Falessi, G. Cantone, S. A. Sarcia, G. Calavaro, P. Subiaco, and C. D'Amore. Peaceful coexistence: Agile developer perspectives on software architecture. *IEEE Software*, 27(2):23–25, 2010.

[47] A. Finkelstein. A Foolish Consistency: Technical Challenges in Consistency Management. In *Database and Expert Systems Applications*, pages 1–5, 2000.

[48] A. Finkelstein, M. Goedicke, J. Kramer, and C. Niskier. Viewpoint Oriented Software-Development - Methods and Viewpoints in Requirements Engineering. *Lecture Notes in Computer Science*, 490:29–54, 1991.

[49] A. Finkelstein, G. Spanoudakis, and D. Till. Managing interference. In *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints' 96) on SIGSOFT'96 workshops*, pages 172–174, 1996.

[50] M. Fowler. Who needs an architect? *IEEE Software*, 20(5):11–13, 2003.

[51] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 9(August):28–35, 2001.

[52] P. Fradet, D. L. Métayer, and M. Périn. Consistency checking for multiple view software architectures. In *Software Engineering—ESEC/FSE'99*, pages 410–428. Springer Berlin Heidelberg, 1999.

[53] M. Galster. Dependencies, traceability and consistency in software architecture: towards a view-based perspective. In *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*, pages 1–4, Essen, Germany, 2011. ACM Press.

[54] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Trans. Software Eng.*, 21(4):1–6, 1995.

[55] J. Garland and R. Anthony. *Large-Scale Software Architecture: A Practical Guide using UML*, volume 9. 2003.

[56] C. Ghezzi and B. Nuseibeh. Special Section-Managing Inconsistency in Software Development-Guest Editorial: Introduction to the Special Section. *IEEE Transactions on Software Engineering*, 24(11):906, 1998.

[57] I. Gorton. *Essential software architecture*. 2006.

[58] I. Graham, B. Henderson-Sellers, and H. Younessi. *The OPEN process specification*. The ACM Press, 1997.

[59] J. Grundy, J. Hosking, and W. B. Mugridge. Inconsistency Management for Multiple-View Software Development Environments. *IEEE Transactions on Software Engineering*, 24(11):960–981, 1998.

[60] R. Haesen and M. Snoeck. Implementing Consistency Management Techniques for Conceptual Modeling. In *Consistency Problems in UML-Based Software Development*, 2005.

[61] M. T. Hansen, N. Nohria, and T. Tierney. What ' s Your Strategy for Managing Knowledge? *Harvard Business Review*, 72(2):106–116, 1999.

[62] F. Harmsen, S. Brinkkemper, and H. Oei. Situational Method Engineering for Information System Project Approaches. In *Methods and Associated Tools for the Information Systems Life Cycle*, number September, pages 169–194, 1994.

[63] A. Hassan and R. Holt. Using development history sticky notes to understand software architecture. In *12th IEEE International Workshop on Program Comprehension*, 2004.

[64] S. J. I. Herzig and C. J. J. Paredis. A conceptual basis for inconsistency management in model-based systems engineering. *Procedia CIRP*, 21:52–57, 2014.

[65] A. R. Hevner. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19(2):87–92, 2007.

[66] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *Management Information Systems*, 28(1):75–105, 2004.

[67] R. Hilliard. Lessons from the Unity of Architecting. In *Software Engineering in the Systems Context*, pages 225–250. 2015.

[68] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, 2007.

[69] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Pearson / Prentice Hall, 2000.

[70] R. Holt. Software architecture as a shared mental model. In *Proceedings of the ASERC Workhop on Software Architecture*, page 64, 2002.

[71] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279, 1997.

[72] Z. Huzar, L. Kuzniarz, G. Reggio, and J. L. Sourrouille. Consistency Problems in UML-Based Software Development. *UML Modeling Languages and Applications*, 3297(1):1–12, 2004.

[73] J. Iivari. A paradigmatic analysis of information systems as a design science. *Scandinavian Journal of Information Systems*, 19(2):5, 2007.

[74] ISO/IEC/IEEE. Standard 42010 - Systems and software engineering — Architecture description. Technical report, 2011.

[75] M. Ivarsson and T. Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, 2011.

[76] S. Jalali and C. Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12*, pages 29–38, New York, New York, USA, 2012. ACM Press.

[77] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings - 5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005*, pages 109–120, 2005.

[78] S. Jansen and S. Brinkkemper. Applied Multi Case Research in a mixed Method Research Project: Customer Configuration Updating Improvement, 2008.

[79] C. N. Johnson. The benefits of PDCA. *Quality Progress*, 35(5):120, 2002.

[80] G. Kaiser and R. Schwanke. Living with inconsistency in large systems. *Proceedings of the International Workshop on Software Version and Configuration Control*, 1988.

[81] R. Kazman, J. Asundi, and M. Klein. Making Architecture Design Decisions: An Economic Approach. Technical report, (CMU/SEI-2002-TR- 035, ADA408740), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[82] R. Kazman and L. Bass. Making architecture reviews work in the real world. *Software, IEEE*, 19(1):67–73, 2002.

[83] R. Kazman, L. Bass, and M. Klein. The essential components of software architecture design and analysis. *Journal of Systems and Software*, 79(8):1207–1216, 2006.

[84] R. Kazman, L. Bass, M. Webb, G. Abowd, and M. Webb. SAAM: a method for analyzing the properties of software architectures. In *Proceedings of 16th International Conference on Software Engineering*, pages 81–90, 1994.

[85] R. Kazman, M. Klein, and P. Clements. ATAM : Method for Architecture Evaluation. Technical report, (No. CMU/SEI-2000-TR-004). Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst., 2000.

[86] D. Kirsh. A Few Thoughts on Cognitive Overload. *Intellectica*, 1(30):19–51, dec 2000.

[87] J. Klein. What Makes an Architect Successful ? *Software, IEEE*, 33(1):20–22, 2016.

[88] K. Knight. Unification: a multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124, 1989.

[89] E. Kornyshova, R. Deneckère, and C. Salinesi. Method chunks selection by multicriteria techniques: An extension of the assembly-based approach. In *Situational Method Engineering: Fundamentals and Experiences*, pages 64–78, 2007.

[90] P. Kruchten. The 4+ 1 view model of architecture. *Software, IEEE*, 12(6):42–50, 1995.

[91] P. Kruchten. An Ontology of Architectural Design Decisions in Software-Intensive Systems. In *2nd Groningen Workshop on Software Variability*, pages 1–8, 2004.

[92] P. Kruchten. What do software architects really do? *Journal of Systems and Software*, 81(12):2413–2416, 2008.

[93] P. Kruchten, P. Lago, and H. van Vliet. Building Up and Reasoning About Architectural Knowledge. In *Quality of Software Architectures*, pages 43 – 58, 2006.

[94] K. Kumar and R. Welke. Methodology engineering: a proposal for situation-specific methodology construction. In *Challenges and Strategies for Research in Systems Development*, pages 257–269, 1992.

[95] P. Lago, P. Avgeriou, and R. Hilliard. Software Architecture: Framing Stakeholder's Concerns. *IEEE Software*, 27(6):20–24, 2010.

[96] C. F. J. Lange and M. R. V. Chaudron. Effects of Defects in UML Models: An Experimental Investigation. In *Proceedings of the 28th International Conference on Software Engineering*, pages 401–411, 2006.

[97] N. Lassing, D. Rijsenbrij, and H. van Vliet. The goal of software architecture analysis: confidence building or risk assessment. *1st BeNeLux Conference on Software Architecture, BENELUX 1999*, pages 47–57, 1999.

[98] T. C. Lethbridge, S. E. Sim, and J. Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10(3):311–341, 2005.

[99] F. J. Lucas, F. Molina, and A. Toval. A systematic review of UML model consistency management. *Information and Software Technology*, 51(12):1631–1645, 2009.

[100] L. Luinenburg, S. Jansen, J. Souer, I. V. D. Weerd, and S. Brinkkemper. Designing web content management systems using the method association approach. In *Proceedings of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, pages 106–120, 2008.

[101] C.-H. Lung, S. Bot, K. Kalaichelvan, and R. Kazman. An Approach to Software Architecture Analysis for Evolution and Reusability. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative Research*, pages 144–154, 1997.

[102] I. Lytra and U. Zdun. Inconsistency Management between Architectural Decisions and Designs Using Constraints and Model Fixes. In *23rd Australian Software Engineering Conference (ASWEC)*, pages 230–239, 2014.

[103] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, 2013.

[104] A. W. Meade and S. B. Craig. Identifying careless responses in survey data. *Psychological Methods*, 17(3):437–455, 2012.

[105] N. Medvidovic, E. M. Dashofy, and R. N. Taylor. Moving architectural description from under the technology lamppost. *Information and Software Technology*, 49(1):2–3, 2006.

[106] K. Mu, W. Liu, Z. Jin, R. Lu, A. Yue, and D. Bell. A merging-based approach to handling inconsistency in locally prioritized software requirements. In *Knowledge Science, Engineering and Management*, pages 103–114, 2007.

[107] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software Reflexion Models: Bridging the gap between Source and High Level Models. *IEEE Transactions on Software Engineering*, 27(4):364–380, 2001.

[108] J. Muskens, R. J. Bril, and M. R. V. Chaudron. Generalizing Consistency Checking between Software Views. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 169–180, 2005.

[109] H. R. Neave. *The Deming Dimension*. SPC Press, Knoxville, TN, 1990.

[110] C. Nentwich. *Managing the Consistency of Distributed Documents*. PhD thesis, University of London, 2004.

[111] C. Nentwich and L. Capra. xlinkit: A consistency checking and smart link generation service. *ACM Transactions on Internet Technology (TOIT)*, 2(2):151–185, 2002.

[112] H. Nissen and M. Jeusfeld. Managing multiple requirements perspectives with metamodels. *IEEE Software*, 13(2):37, 1996.

[113] B. Nuseibeh. To Be and Not to Be : On Managing Inconsistency in Software Development. *8th International Workshop on Software Specification and Design (IWSSD)*, (section 4):164–169, 1996.

[114] B. Nuseibeh. Ariane 5: who dunnit? *IEEE Software*, 14(3):15, 1997.

[115] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software development. *Computer*, 33(4):24–29, 2000.

[116] B. Nuseibeh, S. Easterbrook, and A. Russo. Making inconsistency respectable in software development. *Journal of Systems and Software*, 58(2):171–180, 2001.

[117] H. Obbink, P. Kruchten, and W. Kozaczynski. Software Architecture Review and Assessment (SARA) Report. Technical report, Software Architecture Review and Assessment (SARA) Working Group, 2002.

[118] W. Orlikowski and C. Iacono. Research commentary: Desperately seeking the "IT" in IT research—A call to theorizing the IT artifact. *Information Systems Research*, 12(2):121–134, 2001.

[119] D. L. Parnas. On a 'buzzword': hierarchical structure. In *Pioneers and their Contributions to Software Engineering: Conference on Software Pioneers*, pages 336–339, 1974.

[120] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[121] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[122] D. Pollet and S. Ducasse. Software Architecture Reconstruction : A Process-Oriented Taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.

[123] E. R. Poort. RCDA: Risk- and Cost-Driven Architecture: A Solution Architect's Handbook. Technical report, CGI, 2014.

[124] E. R. Poort and H. Van Vliet. RCDA: Architecting as a risk- and cost management discipline. *Journal of Systems and Software*, 85(9):1995–2013, 2012.

[125] J. Ralyté, R. Deneckère, and C. Rolland. Towards a Generic Model for Situational Method Engineering. In *Advanced Information Systems Engineering*, pages 95–110, 2003.

[126] K. Raymond. Reference Model of Open Distributed Processing (RM-ODP): Introduction. In *Open Distributed Processing*, pages 3–14, 1995.

[127] W. N. Robinson and S. D. Pawlowski. Managing requirements inconsistency with development goal monitors. *IEEE Transactions on Software Engineering*, 25(6):816–835, 1999.

[128] J. Rosik and J. Buckley. Design Requirements for an Architecture Consistency Tool. In *Proceedings of the 21st Annual Psychology of Programming Interest Group Conference*, pages 109–124, 2009.

[129] N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. 2009.

[130] N. Rozanski and E. Woods. Applying Viewpoints and Views to Software Architecture. Technical Report 2005, http://www.viewpoints-and-perspectives.info/vpandp/wp-content/themes/secondedition/doc/VPandV_WhitePaper.pdf, 2011.

[131] J. Rudd, K. Stern, and S. Isensee. Low vs. High-Fidelity Prototyping Debate. *Interactions*, 3(1):76–85, 1996.

[132] F. Samer and S. Lee. Coordinating Expertise in Software Development Teams. *Management Science*, 46(12):1554–1568, 2000.

[133] K. Schwaber. SCRUM Development Process. In *Business Object Design and Implementation*, pages 117–134, 1997.

[134] M. Shaw and P. Clements. The Golden Age of Software Architecture: A Comprehensive Survey. *IEEE Software*, 23(2):31–39, 2006.

[135] Y. Shinkawa. Inter-model consistency in UML based on CPN formalism. In *13th Asia Pacific Software Engineering Conference (APSEC)*, pages 411–418, 2006.

[136] H. A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.

[137] J. Singer and T. Lethbridge. An Examination of Software Engineering Work Practices. In *CASCON First Decade High Impact Papers*, pages 174–188, 2010.

[138] G. Spanoudakis, A. Finkelstein, and D. Till. Overlaps in requirements engineering. *Automated Software Engineering*, 6(2):171–198, 1999.

[139] G. Spanoudakis and A. Zisman. *Inconsistency Management in Software Engineering: Survey and Open Research Issues*, volume 1. 2001.

[140] S. M. Sutton and I. Rouvellou. Concern Modeling for Aspect-Oriented Software Development. *Aspect-Oriented Software Develpoment*, 21(1):755, 2004.

[141] A. Tang, M. A. Babar, I. Gorton, and J. Han. A survey of architecture design rationale. *Journal of Systems and Software*, 79(12):1792–1804, 2006.

[142] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6):918–934, 2007.

[143] A. Tang and M. F. Lau. Software architecture review by association. *Journal of Systems and Software*, 88(1):87–101, 2014.

[144] A. Vallecillo. RM-ODP: The ISO Reference Model for Open Distributed Processing. *DINTEL Edition on Software Engineering*, 3:pp. 69–99, 2001.

[145] I. van de Weerd and S. Brinkkemper. Meta-Modeling for Situational Analysis and Design Methods. In *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*, chapter 3, pages 38–58. Idea Group Publishing, 2008.

[146] I. van de Weerd, S. Brinkkemper, J. Souer, and J. Versendaal. A Situational Implementation Method for Web-based Content Management System-applications: Method Engineering and Validation in Practice. *Software Process Improvement and Practice*, 11(4):361–371, 2006.

[147] M. van Harmelen. Exploratory user interface design using scenarios and prototypes. In *People and computers V: proceedings of the fifth conference of the British Computer Society Human-Computer Interaction Specialist Group, University of Nottingham*, page 191, 1989.

[148] V. Venkatesh, M. Morris, G. Davis, and F. Davis. User acceptance of information technology: Toward a unified view. *MIS Quarterly*, pages 425–478, 2003.

[149] G. Walsham. *Interpreting information systems in organizations*. Chichester: Wiley, 1993.

[150] A. Wasserman. Toward a Discipline of Software Engineering. *IEEE Software*, 13(6):23, 1996.

[151] R. Wieringa. Design Science as Nested Problem Solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, page 8, 2009.

[152] C. Wohlin. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 38, 2014.

[153] E. Woods. Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report. In *Software Architecture*, pages 182–193, 2004.

[154] E. Woods. Architecture Description Languages and Information Systems Architects : Never the Twain Shall Meet ? Technical report, Artechra Whitepaper, 2005.

[155] E. Woods. Industrial architectural assessment using TARA. *Journal of Systems and Software*, 85(9):2034–2047, 2012.

[156] E. Woods and R. Hilliard. Architecture description languages in practice session report. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 243–246, 2005.

[157] T. Yang and C. W. Chen. An incentive pay system for project management based on responsibility assignment matrix and fuzzy linguistic variables. *Expert Systems with Applications*, 36(10):12585–12591, 2009.

[158] R. K. Yin. *Case Study Research: Design and Methods*. Sage Publications Inc., 3 edition, 2003.

[159] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann. Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method. In *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 157–166, 2008.

Appendices

Consistent Inconsistency Management: a Concern-Driven Approach

Jasper Schenkhuizen[1], Jan Martijn E.M. van der Werf[1], Slinger Jansen[1], and Lambert Caljouw[2]

[1] Department of Information and Computing Science
Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{j.schenkhuizen, j.m.e.m.vanderwerf, slinger.jansen}@uu.nl
[2] Unit4
Papendorpseweg 100
3528 BJ, Utrecht, The Netherlands
lcaljouw@unit4.com

Abstract. During the development of a software system, architects deal with a large number of stakeholders, who need to work collaboratively from multiple perspectives. This inevitably leads to inconsistency: goals, concerns, design decisions, and models are interrelated and overlapping. Existing approaches to support inconsistency management are limited in their applicability and usefulness in day to day practice due to a lack of documentation and the presence of incomplete, informal and heterogeneous models in software architecture.
This paper presents a lightweight generic method, the Concern-Driven Inconsistency Management Method (CDIM), that aids architects with inconsistency management in software architecture. Evaluation in two case studies shows that the CDIM method provides an effective way to explicate and structure the sometimes ad-hoc and implicit process of inconsistency management that software architects currently follow. The method also shows to be a good instrument for tradeoff analysis, guiding the architecture definition process.

1 Introduction

Inconsistency is prevalent and ubiquitous in software development and software architecture (SA) [7]. Although inconsistency in software architecture is not necessarily a bad thing [22], undiscovered inconsistency leads to all kinds of problems [21, 25]. Notable examples include the disastrous mission failures in aeronautics [24].

Inconsistency is present if two or more statements made about the system or its architecture are not jointly satisfiable [9], mutually incompatible, or conflicting [3]. Various examples of inconsistencies exist: failure of a syntactic equivalence test, nonconformance to a standard or constraint [9], two specified incompatible requirements, or two developers implementing a non-relational and a relational database technology for the same database, to name a few.

Inconsistency arises inevitably due to the fact that SA is concerned with heterogeneous, multi-actor, multi-view and multi-model activities [23]. Finkelstein [5] argues

that a major cause of inconsistency is the fact that multiple actors need to work together, each with different views, opinions and interpretations. Different stakeholders have different languages or development strategies, addressing different stages of the development, having partly overlapping areas of concern, or having different technical or economic objectives [23].

Consequently, this heterogeneity and the diverse context of SA causes inconsistency management to be inherently difficult [25]. In addition, a lot of architectural knowledge is contained in the heads of involved architects and developers [16] and documentation of architectural design decisions (ADDs) and the architecture design is often not maintained over time [11, 27]. Available architectural information can be specified in a wide range of formats such as ad-hoc informal drawings, semi-structured text documents, or more formal models. Though inconsistency management is needed, the possibilities for managing inconsistency in software architecture are limited [21], and architects thus benefit from methods that aid in management of inconsistency.

Traditional approaches are based on logic or model-checking. The former uses formal inference techniques to detect model inconsistency, which makes them difficult to scale. The latter disposes model-verification algorithms that are sufficiently suited to detect specific inconsistencies, but do not fit well to other kinds of inconsistency [2]. Additionally, those approaches often require preliminary work in the form of formalization of models. Currently, no appropriate infrastructure is available that is capable of managing a broad class of inconsistency [9].

To address the limitations of related approaches and to support the architect in the difficult process of inconsistency management, this paper proposes a simple, lightweight method, enabling the architect to systemathically manage inconsistency: The Concern-Driven Inconsistency Management (CDIM) method. CDIM identifies important concerns of different stakeholders, as these form an source of inconsistency [23], and utilizes a matrix to discuss overlapping concerns to find and manage diverse types of inconsistency. CDIM consists of a 7-step cyclic process, based on the Plan Do Check Act (PDCA) cycle [12], work of Nuseibeh [25], Spanoudakis [30], and related architecture evaluation methods.

It is cost-effective to discover inconsistency early in the process [9]. One of the first elements that an architect needs to consider, are *concerns* [10]. Furthermore, concerns are the driving force of building an architecture and designing the system [17]. Software architects benefit from inconsistency discovery and management at an initial stage, as principal decisions are often hard to reverse. Conflicting decisions are a source of inconsistency in SA, which emphasizes the need for managing both model inconsistency as well as inconsistency in concerns and design decisions.

The remainder of this paper is structured as follows: Sec. 2 illustrates a brief example used to explain the CDIM method. Sec. 3 provides a short overview of inconsistency management in practice. The CDIM method is demonstrated in Sec. 4, followed by the results of the evaluation (Sec. 5), and the discussion, conclusion and directions for future work in Sec. 6 and Sec. 7.

2 Running Example: USG

As a running example throughout the paper, we present the USG-system, which is based upon a real-life scenario that emerged during one of the case studies. The domain and company names are fictive due to non-disclosure agreements. A new system must be built for the Union of Security Guards (USG). A new law states that every security guard in the Netherlands should receive a Securibadge (SB) that every security guard should wear during shifts. Initial brainstorming sessions show that the system should consist of two subsystems, indicated in blue. A central system (A) should be developed, which is a portal that registers a new badge application, and loads the unique identifier of the SB into a central data store in the Netherlands. Every security guard in the Netherlands should be registered at system A. On the other hand, an on-site check-in-and-out system B should be built, which is used at every event site, so that guards can register their shifts. Thus multiple systems B communicate with a single system A. There are 70.000 nightly security guards in the Netherlands, of which 95% start their shift between 10pm and 11pm. In other words, system B has to deal with 66.500 check-ins within one hour. Furthermore, the USG has stated that the system should store the check-in data for at least 5 years. That equals a large amount of data: 66.500 guard ID's, times 5 years, times 220 days in a year, times 2 for checking in and out. A contextual overview of the systems is shown in Fig. 1.

A challenging factor is that most employees work for the large company *Secunite*. It has its **own clock-in systems** at many venues and event sites. Thus, many security guards will clock-in using the *Secunite* system, instead of system B developed for the USG. Therefore, system B should also be able to accept check-ins from the *Secunite* clock-in system, and match the external clock-in with the respective Securibadge of the security guard.

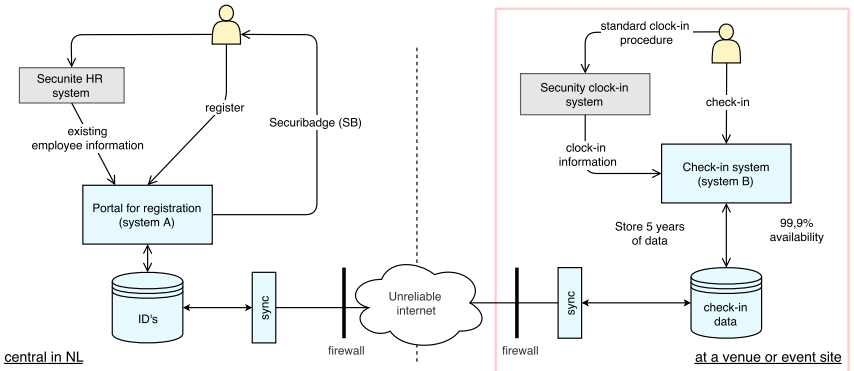


Fig. 1. Contextual overview of the USG systems. Floating text refers to certain constraints, text on the arrows refer to actions or information.

3 Inconsistency management in SA

An important task of the software architect is inconsistency management (IM): identifying inconsistency, preserving it when acceptable, and deferring or solving it when required [5, 25]. Spanoudakis and Zisman [30] propose a framework for IM, based upon [6] and [25], consisting of six activities: (1) *detection of overlaps*, (2) *detection of inconsistencies*, (3) *diagnosis of inconsistencies*, (4) *handling of inconsistencies*, (5) *tracking of findings*, and (6) *specification of an IM policy*. A critical component in IM is identifying overlap, as it is a precondition for inconsistency [6]. Overlap in concerns occur when associated design decisions influence each other, and emerges due to different views, assumptions, and concerns all being interrelated because they are related to the same system [21]. The presence of such interrelations introduces the potential for inconsistency [30]. Techniques that focus on *detection of overlaps* do this based on for example representation conventions, shared ontologies, or similarity analysis [30]. Techniques for *detection of inconsistencies* are logic-based approaches, model-checking approaches, specialized model analyses, and human-centered collaborative approaches [30]. Inconsistency diagnosis is concerned with the identification of the source, cause and impact of an inconsistency [30]. Handling inconsistency is concerned with the identification and execution of the possible actions for dealing with an inconsistency. Tracking refers to recording important information of the inconsistency in a certain knowledge base [30].

Inconsistency in SA has a wide range of dimensions, such as inconsistency in code [29], inconsistent requirements or concerns [32], and model inconsistency [2]. We consider these to be 'tangible' forms of inconsistency. An 'intangible' inconsistency is often denominated as a *conflict*, still being undocumented or unspecified. A conflict between concerns occurs if their associated design decisions are mutually incompatible, or negatively affect each other. That is, a conflict has the potential to manifest itself as an inconsistency, due to designs that follow from design decisions, in their turn caused by design concerns [32]. Thus, if design decisions are *conflicting* (intangible inconsistency), they are mutually incompatible [3], and can lead to tangible inconsistency.

This corresponds with the view to see architectural design decisions as first-class entities [17]. Adopting the definition that a software system's architecture is the set of principal design decisions about the system [3], we see that inconsistency – even though it may be intangible (undocumented) at early stages – already emerges if design decisions are inconsistent or contradictory. Intangible inconsistency could manifest itself in tangible model inconsistency or code inconsistency for example. At early stages in the architecting process, architects deal with coarse-grained models and high-level design decisions, which are usually recorded and documented using more informal notations [3]. As a result, related formal and model-checking approaches for inconsistency management are less applicable.

Approaches that are more applicable in this context are stakeholder-centric methods for inconsistency management. These involve human inspection of overlap, and human-based collaborative exploration [30] (such as Synoptic [4] or DealScribe [26]). These techniques assume that detection of inconsistency is the result of a collaborative inspection of several models by stakeholders [30]. Approaches such as Synoptic [4] and DealScribe [26] solely focus on model inconsistency, and therefore, cannot be used for

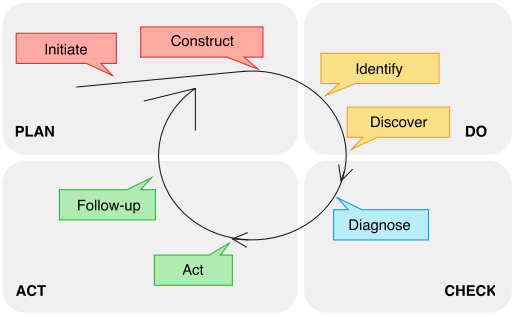


Fig. 2. The CDIM cycle.

other types of inconsistency, such as inconsistent design decisions. Synoptic requires stakeholders to specify conflicts in so-called 'conflict forms' to describe conflicts that exist in models. In DealScribe, stakeholders look for 'root-requirements' in their models. Root requirements are identified for concepts present in the models, and pairwise analysis of possible interactions between root requirements results in a list of conflicting requirements. A limitation of this approach is that pairwise sequential comparison is time-consuming and labour-intensive.

IM is a part of the process of verification and validation [9]. Verification and validation of an architecture is done with the use of architectural evaluation methods (AE) methods. Several AE methods are developed over the past decade, and many of them have proved to be mature [1]. Therefore, they have formed a source of inspiration for the CDIM method. Fragments of the following methods have been included in the development of the CDIM method: SAAM, ATAM, APTIA, ALMA, SAAMER, SBAR, and TARA. Due to space limiations, we refer the reader to e.g. [1] for a discussion on and comparison of various AE methods.

4 Concern-Driven Inconsistency Management

In this paper, we propose the Concern-driven Inconsistency Management (CDIM) method to systematically identify and to keep track of inconsistency, based on concerns and perspectives. CDIM is developed using the Method Association Approach (MAA) [19] together with input from experts through semi-structured interviews and smallweight case studies. MAA takes existing methods into account to methodically assemble a new method for use in new situations [19]. CDIM is based on the process of inconsistency management as defined by Spanoudakis [30] and Nuseibeh [25], combining several SA evaluation and inconsistency management techniques with the traditional iterative phases *Plan*, *Do*, *Check*, and *Act* (PCDA) cycle [12]. These 4 phases are divided in 7 activities, depicted in Fig. 3. Each of the 7 activities contain multiple steps. In the remainder of this section, we present these 7 activities divided over the 4 phases in detail.

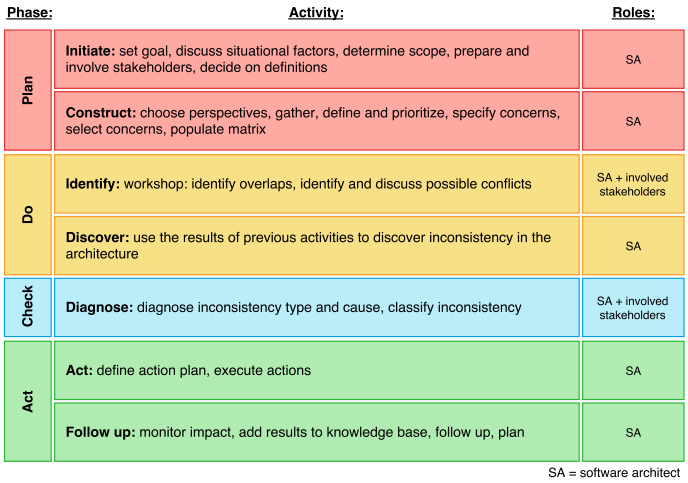


Fig. 3. CDIM practices.

4.1 Plan Phase

The use of templates to capture information makes methods more consistent across evaluators [13]. To ease IM, we propose the use of concern cards, a template that enables reasoning about a concern. Such a template makes methods more consistent across evaluators [13]. Additionally, the use of cards in software development is not unusual (such as planning poker [8] used by many SCRUM teams [18]). A concern card assists the architect in collecting and understanding the different concerns, by making these explicit. A concern card consists of:

1. a unique identifier,
2. a short, concise name,
3. a comprehensive definition and explanation of the concern,
4. the concern's priority,
5. related stakeholders that have an interest in the concern,
6. the perspective or category to which the concern belongs, and
7. possible associated architectural requirements, which can be used during discussion.

The Plan phase consists of two activities to (1) develop an action plan, and (2) to collect and select the necessary concerns and perspectives. This phase results in an action plan, a concern-card desk, and a prioritized matrix of concerns.

Initiate. In this activity, the architect develops an action plan containing the goal of the CDIM cycle, the scope of the architecture under analysis, the organization's situational factors, and which stakeholders are needed during the CDIM cycle. Together with the main stakeholders, the architect defines the focus areas that need to be addressed, and

decides upon a shared vocabulary. The outcome of this step is an action plan and a general overview of the important areas of concern.

Construct. During this activity, the architect selects perspectives and collects concerns from stakeholders. A perspective enables the architect to categorize the concerns gathered and to analyze the architecture from a certain angle. For each perspective, the architect collects important and relevant concerns. Concern cards are created by specifying and prioritizing the identified concerns based on their relative importance. Important is that prioritization is revisited frequently during the CDIM cycle. Next, the architect constructs a matrix and populates the matrix' axes with the specified concern cards of two perspectives. The main outcome of this activity is a deck of concern cards together with the matrix.

In our running example, the goal is to identify inconsistent design decisions, and the scope is indicated by a red rectangle (Fig. 1). The stakeholders to be involved are the developers, the contractor, and the USG. In the *construct* activity, the architect selects 2 perspectives and 4 concerns. The security perspective contains the concerns 'protect data leakage' and 'prevent fraudulent check-ins'. The performance perspective contains 'deal with peak load' and 'low response time of system'. The matrix is represented in Fig. 5.

4.2 Do Phase

The next phase consists of two activities: identify and discover. The former tries to identify possible conflicts through a workshop with the relevant stakeholders. Based on these possible conflicts, the architects, possibly together with stakeholders, go through the existing architecture to discover whether these are actual inconsistencies.

Identify. This activity is a workshop with architect and important stakeholders to identify possible conflicts. Input is the previously constructed matrix. The principal idea behind the cells of the matrix is that these provide the hotspots: areas in the architecture where concerns could possibly overlap or conflict. Multiple overlaps or conflicts may be contained in each cell, as visualized in Fig. 4. The "hotspots" are discussed by the architect and stakeholders. Their role is to aid the evaluator with deciding on how conflicts affect the architecture or possibly could affect the architecture, and which conflicts could be problematic. This should always be done in line with the project goal. We envision the workshop to entail the following steps:

1. Explain the goal of the workshop;
2. Explain the matrix, its axes and cells;
3. For each cell:
    a identify importance;
    b identify overlap and possible conflicts;
4. Summarize results;

The main outcome is a completed matrix, presenting the architect with important conflicting concerns, indicating places where inconsistencies may arise.

Discover. In case an architecture already exists, the architect uses the matrix to find possible inconsistent specifications on the basis of the identified conflicts. The inputs of this activity are the completed matrix, an architecture design rationale, and the architecture (documentation) itself. The architect should develop a design rationale if the design rationale is non-existent. The architect uses a combination of his expertise and knowledge of the system, to systematically search for important inconsistencies, using the conflicts identified in the matrix. Given the deliberate simplicity of CDIM, and the complexity of a software architecture context, the steps in this phase are inevitably one of judgment rather than quantifiable assessment. The drawback is that this approach is less explicit and more based on subjective factors as intuition and experience. The advantage is that this is easier and more flexible. The main outcome of this activity is a list of important inconsistencies in the architecture.

In our running example, the Identify activity leads to an identified conflict between 'prevent fraudulent check-ins' and 'deal with peak load'. Preventing fraudulent check-ins in system B (Fig. 1) is addressed by only accepting certified check-ins, which contradicts the concern that the system should deal with the peak load of 66.500 check-ins between 10pm and 11pm, so they are in conflict. During the next activity, the architect discovers an inconsistency between the design, that states that system B will check 'certificates' when check-ins are entered in the system, and the peak load, that cannot be dealt with if the system must check all certificates.

4.3 Check Phase

Once the matrix is completed, and possible inconsistencies in the architecture have been discovered, the architect classifies the inconsistency and determines the type and cause of the inconsistency. This is done in the next activity:

Diagnose. Inconsistencies have several types and abstraction levels [25, 30]. Examples are *model inconsistency* [2], inconsistent requirements, or inconsistencies on code level [23]. Each type has several subclasses. In this activity, the architect identifies the type, and subsequently searches for the possible causes. Next, the architect classifies the inconsistency. Classification is done on the basis of four aspects: *impact*, *business value*, *engineering effort*, and the individual *characteristics* of the inconsistency (such as the type and cause). 'Impact' refers to an inconsistency's consequences, 'Business value' to whether the inconsistency is perceived as critical by the business, 'Engineering effort' addresses the effort of solving an inconsistency and the availability of design alternatives, and 'Characteristics' addresses factors related to the inconsistency itself such as the type or cause of the inconsistency. These factors are context-specific but should be considered as well [25]. The output is a *classification* in terms of these factors. Results from the matrix could be visualized as presented in Fig. 4.

The discovered inconsistency in our running example is a model inconsistency. The cause is incomplete knowledge of the architect about other conflicting concerns, and the corresponding classification is (*high,high,medium,low*). The inconsistency, when undetected, has a high impact on the system once it would be implemented, hence a

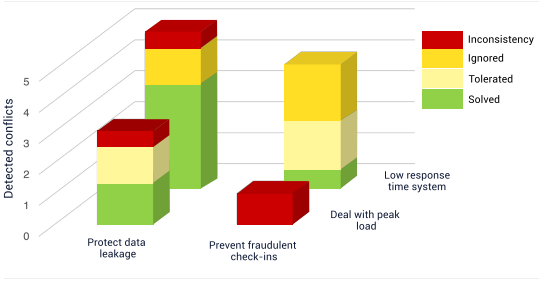


Fig. 4. The architect can use an overview of the amount of inconsistencies that are still open (red), that are carefully ignored (bright yellow), tolerated (cream yellow), or that have been solved (green) to indicate what needs to be done.

high business value, a medium engineering effort (it can pretty easily be solved), and the characteristics are not considered, as the impact is high.

4.4 Act Phase

In the Act phase, for each of the diagnosed inconsistencies an action plan is created, together with a follow-up of CDIM itself. This leads to the following two activities:

Act. For each diagnosed inconsistency, the architect evaluates its impact and defines an action plan for settling the inconsistency. Based on interviews with software architects and earlier work of [25], we identify five actions for settling inconsistencies: 'resolve', 'ignore', 'postpone', 'bypass', and 'improve'. It is important to note that handling an inconsistency is always context-specific and requires human insight.

'Resolving' the inconsistency is recommended if the impact is high, the business value is high, regardless of the engineering effort. Solving an inconsistency could be relatively simple (adding or deleting information from a description or view). However, in some cases resolving the inconsistency relies on making important design decisions (e.g. the introduction of a complete new database management technology) [6, 25]. 'Ignoring' the inconsistency is recommended if the potential impact is low, the business value is low, and the engineering effort is high. 'Postponing' the decision is recommended when both impact and business value are relatively low, and engineering effort is relatively high. Postponing provides the architect with more time to elicit further information [25]. 'Bypassing' is a strategy in which the inconsistency is circumvented by adapting the architecture in such a way that the inconsistency itself still exists, but not touched upon. It is recommended when the current impact is low and the business value and engineering effort are relatively high, in which case continuity is important. 'Improving' on an inconsistency might be cost-effective in other situations, in which time pressure is high, and the risk is low. To improve on an inconsistency, inconsistent models can be annotated with explanations, in order to alleviate possible negative consequences of the inconsistent specification. Best practices have shown that it is wise to revisit the action plan when a system evolves [25]. The main outcome are the actions

referring to how and when to settle the inconsistency, possibly with requests for change on the architecture, code or any other specifications that are important.

Follow up. In the final activity of CDIM, the architect assesses the impact of the actions, adds gained knowledge to the architectural knowledge base, and determines how to proceed. To assess the impact, the architect should check (a) whether an action intervenes with other actions; (b) whether a handling action affects existing concerns; and (c) whether a handling action results in new concerns. The architect decides whether a new cycle is needed, or iteration to previous steps is needed.

In our running example, the inconsistency must be solved. The architects add a delay to system B: by accepting all check-ins during the peak load and checking the certificates afterwards, system B can undo and reverse fraudulent check-ins in hindsight, while not hindering the performance during peak load. As a follow-up, a new cycle can be started to check the concerns and perspectives again after initial implementation.

5 Evaluation

We have evaluated CDIM through two case studies and two semi-structured expert interviews. The case studies consisted of a documentation study beforehand, executing a limited instance of CDIM, and a documentation study afterwards, where limited refers to a limited amount of concerns, a virtual involvement of stakeholders, and a small part of an architecture.

5.1 Method

In the semi-structured interviews, experts performed a walkthrough of CDIM. Participants in the case studies are referred to as case experts and participants of the interviews are referred to as interview experts. Detailed information about the participants is presented in Tbl. 1. After the sessions, all experts were presented with statements (Tbl. 2) and questions assessing the strengths and weaknesses of the CDIM method, the perceived usefulness, and the intention to use. Transcripts and possible questions were discussed through e-mail. The participants were asked to indicate whether they would agree with the statement or not, using a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree, 3 = neutral). Statements marked with a black downwards triangle were formulated negatively in order to avoid inattentiveness in the responses [20]. Resulting insights from the questions on the strengths and weaknesses, perceived usefulness and intention to use are discussed in the following section (Sec. 6).

Table 1. The background information on participating experts of the evaluation.

#	Type	Org. size	Org. type	Job title	SA exp.
1	Case study	Large (2.000+)	IT Services	Principal IT architect	20 years
2	Case study	Large (700+)	Online retail	Lead IT architect	07 years
3	Expert Interview	Large (2.000+)	IT Services	Software architect	25 years
4	Expert Interview	Small (50+)	Healthcare IT	Software architect	10 years

(SA = software architect)

Table 2. The statements and responses of the case experts and the interview experts.

		Statement	Case 1	Case 2	Interv. 1	Interv. 2
	1	The guidelines and process support are clear.	3	4	4	4
	2	It is clear to me in which stage the method should be applied.	5	5	5	5
	3	I have a technique for each step if it is required.	5	4	4	5
▼	4	It is not feasible to identify and involve stakeholders in this process.	1	1	1	1
	5	A design rationale and a knowledge base promote reusability.	3	3	5	4
	6	The method fits in my daily practices as a software architect.	4	4	3	4
▼	7	The method feels restrictive and compelling.	2	1	1	1
	8	The method does not constrain my architectural freedom.	5	4	4	5
▼	9	The method is difficult.	3	2	3	1
	10	The method is flexible.	4	4	4	5
	11	Concerns are an effective way to detect inconsistency.	4	3	4	4
	11	Using a matrix is an effective way of detecting overlaps.	4	5	4	5

(1=strongly disagree, 5=strongly agree, 3=neutral)

5.2 Results

Experts' appreciations of the statements are shown in Tbl. 2. In the text, statements are referred to as (#n). The numbers in the rightmost four columns represent the Likert scores the experts expressed. All experts appreciated the guidelines and process support (#1). Experts indicated that it was clear how CDIM can be applied and embedded, and suggested to "*use the method anytime when new concerns arise*". Experts argued that stakeholder involvement is desirable at all times, however not feasible in all contexts (#4). Case experts adopted a neutral stance towards the effectiveness of a design rationale and a knowledge base as means to promote reuse in IM. The CDIM method was found to be a good fit in the daily practices of the experts (#6,7): "*the CDIM structures the steps that are implicitly taking place in my head*". However, not all steps were received equally positively. The first phase (Plan) was received more restricting than the Do, Check, and Act. CDIM does not constrain the architect's architectural freedom: "*it adds structure to the process at most"* (#8). Experts adopted a positive attitude towards the flexibility and difficulty of CDIM (#10). Concerns were found reasonably effective constructs to detect overlaps as some experts "*found it difficult to express appropriate names for concerns*", while they were "*already thinking of a solution for conflicting concerns*" (#11). Using the CDIM matrix appeared to be an effective way to detect conflicts (#11).

5.3 Evaluation of the Running Example: USG

Executing CDIM in the USG case study resulted in a matrix of 2 perspectives: *performance* and *security*, each containing two concerns, which are illustrated in Fig. 5. The architect did not have a documented design rationale while he already made several decisions on the system's architecture. 'Preventing data leakage' is addressed by

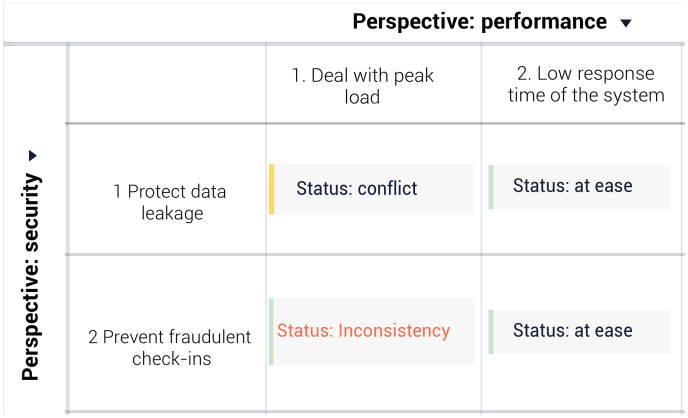


Fig. 5. An excerpt of the CDIM matrix, used in the running example USG.

encrypting the entire database. By issuing certificates for each system that is connected with system B, and only accepting certified check-ins, the architect expected to address 'fraudulent check-ins'. The architect identified two conflicts between the concerns in the matrix, indicated in the leftmost cells (Fig. 5). "*Certification to prevent fraudulent check-ins has consequences for the performance of the system.*" It means that it will be costly and time-consuming to check all the certificates. This concern contradicts 'deal with peak load', as the architect stated it: "*The matrix shows this. I can solve this by accepting every check-in during the peak hours, and afterwards check whether all certificates are authorized or not.*" CDIM explicated conflicting design decisions that the architect otherwise would have made implicitly: "*the method prevented me from making inconsistent design decisions.*"

On the basis of this case we could not confirm that an execution of the CDIM cycle resulted in detected inconsistencies in the architectural description, as there was little to no documentation available. Moreover, both case studies demonstrated that defining concerns led to confusion among stakeholders and architects. If the concerns and perspectives are defined on the wrong level, the method might not be applicable or produce inaccurate results. If concerns are too high level, it might be difficult to associate design decisions with them, while those connections between design decisions and concerns appeared to be very critical for finding inconsistencies or conflicts. In general, we observed that the participants came to helpful insights regarding the architecture, and the CDIM method helps the architects in identifying tradeoffs through the use of concerns.

6 Discussion

Assessing whether CDIM actually results in detected inconsistency is difficult, as a lot of other factors influence detected inconsistencies in architectural documentation: for example the strategies and languages of developers that make and maintain models [23], or the experience of the software architect that searches for inconsistency – it

remains a qualitative practice. Results from the case studies and expert interviews led to interesting findings.

The method itself yields practical results in the form of design solutions, instead of pure descriptive results such as how many inconsistencies have been detected. This enables the CDIM matrix to be used as a design rationale. Tang [31] illustrated that software architects often do not document design rationale as they have no time or budget, or no suitable tool at their disposal. Architects struggling with documenting a design rationale can benefit from this when using CDIM.

Furthermore, a lot of steps that architects would usually perform implicitly, were explicated by our method. Explicating these conflicts and inconsistencies makes the process less error-prone. Experts argued that they appreciated that CDIM "really structures" the process and forces the architects to deal with inconsistency management consciously: "*If correctly executed, the method could help identifying conflicts and inconsistencies earlier and can thus prevent rework*".

While CDIM may not be optimal for detecting inconsistencies in documentation or models, CDIM method is especially useful for identifying tradeoffs and other conflicts through the use of concerns and the matrix. This might indicate an increase in the architect's productivity, which contributes to the usefulness of method. Participants support this by indicating they 'came to design solutions and ideas'. The matrix is perceived as a comprehensive way to detect overlapping and conflicting concerns, given that the right concerns are selected. Another strength of CDIM is that its modularity enables the architects to use the method with a 'toolkit approach' [28], referring to the fact that architects use those elements of the toolkit they need and ignore the rest. The lightweight and flexible character of CDIM was perceived positively during the evaluation.

The evaluation shows that involvement of relevant stakeholders is desired, but not always feasible. While involvement of stakeholders for analyzing an architecture introduces extra 'weight' to a method [1], stakeholder-centric and human-based approaches are the only method for detecting inconsistency in informal models or models expressed in different languages [30].

Furthermore, even though architecture evaluation methods such as the Scenario-based Architecture Analysis Method (SAAM) [14] and the Architecture Tradeoff Analysis Method (ATAM) [15] have a different direct goal than our method, they are related. Different from these, CDIM can be applied *during* the architecting process, as it does not require the architectural design to be present [1]. CDIM provides alleviation of the resources required for performing the ATAM and SAAM. The reason for this is that conflicts may possibly already be discovered or prevented in an earlier phase of the design. In this way, CDIM can be used complementary to an organization's existing AE methods.

Overall, the experts found the CDIM method primarily useful for detecting intangible inconsistency in the form of conflicts and inconsistent design decisions. This is a strength, because incompatible or conflicting decisions can be a source of problems in the final product [32]. Participants of the case studies indicated to have an intention to use the CDIM method. We noticed that experts preferred usage of the matrix (Do and Check), while being less enthusiastic on building the matrix (Plan). Several improve-

ments were suggested, such as an 'initial fill' of the matrix, offering architects explicit examples of a populated matrix with concerns and perspectives.

6.1 Threats to Validity

Since the case studies had certain limitations (limited time, involved stakeholders and concerns), it is difficult to infer whether the participants' insight had really increased. Furthermore, experts that participated in the development of CDIM were part of the evaluation as well. This may have caused a threat to the internal validity of this research, as experts' prior knowledge of the CDIM method might have influenced the results. Further, as CDIM is a generic method, it requires taylorization to be applicable in an organization. Moreover, even though several authors underline the importance of concern-centric architecting [10] and viewing concerns as primary drivers of software architecture [28], the effectiveness of using concerns for detecting *tangible* inconsistency is debatable. Nonetheless, the evaluations indicate that concerns appear to be useful for identifying overlaps and conflicts. As our method is partly built upon expert knowledge, caution is needed on generalizing the results.

7 Conclusions & Future work

Managing inconsistencies in software architecture consistently and systematically is a difficult task. This paper presents CDIM, an inconsistency management method that supports the software architect in managing and detecting inconsistencies and conflicts in software architecture. Through concern cards, the architect is able to discover and document the relevant concerns. With the use of the CDIM matrix, the architect identifies overlapping and conflicting concerns. In this way, CDIM enables architects to detect (undocumented) inconsistent design decisions as well as it helps architects to search for inconsistency in informal models. Initial evaluation through two case studies and two semi-structured interviews, shows that the method is appreciated by the participating experts to make inconsistencies tangible, and to support them in the analysis and evaluation of the architecture. Considerable benefits of the CCIM method resulting from the evaluation are its simplicity, flexibility and the applicability of the method, in combination with fast, practical results in the form of architectural design solutions. Another feature of the method is that the resulting CDIM matrix can be used as a tool for communication and as a design rationale for motivating design decisions, thereby being an instrument for architecture tradeoff analysis as well.

In terms of directions for future research, further work could address building a functional prototype and validating it in practice. Other areas of further research should determine the effectiveness of CDIM on a larger scale. More research on the use of concerns as first-class entities in this domain would help us to establish a greater degree of accuracy on this matter. We hope that we have presented a new refreshing 'human' look upon the topic of inconsistency management in software architecture. We feel that this is a welcome input in the contemporary context of software architects, as eventually it are the stakeholders and their concerns that are so crucially important for this domain.

References

1. M. A. Babar, L. Zhu, and R. Jeffery. A framework for classifying and comparing software architecture evaluation methods. In *15th Australian Software Engineering Conference (ASWEC 2004)*, pages 309–319. IEEE Computer Society, 2004.
2. X. Blanc, I. Mounier, A. Mougenot, and T. Mens. Detecting model inconsistency through operation-based model construction. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 511–520. ACM, 2008.
3. E. M. Dashofy and R. N. Taylor. *Supporting Stakeholder-driven, Multi-view Software Architecture Modeling*. PhD thesis, 2007.
4. S. Easterbrook. Handling conflict between domain descriptions with computer-supported negotiation. *Knowledge Acquisition*, 3(3):255–289, 1991.
5. A. Finkelstein. A foolish consistency: Technical challenges in consistency management. In *Database and Expert Systems Applications, 11th International Conference, DEXA 2000*, volume 1873 of *LNCS*, pages 1–5. Springer, Berlin, 2000.
6. A. Finkelstein, G. Spanoudakis, and D. Till. Managing interference. *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops* -, (MARCH 1999):172–174, 1996.
7. C. Ghezzi and B. Nuseibeh. Guest editorial: Introduction to the special section - managing inconsistency in software development. *IEEE Trans. Software Eng.*, 25(6):782–783, 1999.
8. J. Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3, 2002.
9. S. J. I. Herzig and C. J. J. Paredis. A conceptual basis for inconsistency management in model-based systems engineering. *Procedia CIRP*, 21:52–57, 2014.
10. R. Hilliard. Chapter 10: Lessons from the Unity of Architecting. In *Software Engineering in the Systems Context*, pages 225–250. 2015.
11. A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Fifth Working IEEE / IFIP Conference on Software Architecture (WICSA 2005)*, pages 109–120. IEEE Computer Society, 2005.
12. C. N. N. Johnson. The benefits of PDCA. *Quality Progress*, 35(3), 2002.
13. R. Kazman, L. Bass, and M. Klein. The essential components of software architecture design and analysis. *Journal of Systems and Software*, 79(8):1207–1216, 2006.
14. R. Kazman, M. Klein, L. J. Bass, M. Webb, and G. D. Abowd. SAAM: A method for analyzing the properties of software architectures. In *Proceedings of the 16th International Conference on Software Engineering*, pages 81–90. IEEE Computer Society / ACM Press, 1994.
15. R. Kazman, M. Klein, and P. Clements. ATAM : Method for Architecture Evaluation. Technical report, 2000.
16. P. Kruchten, P. Lago, and H. van Vliet. Building up and reasoning about architectural knowledge. In *Second International Conference on Quality of Software Architectures, QoSA 2006*, volume 4214 of *LNCS*, pages 43–58. Springer, Berlin, 2006.
17. P. Lago, P. Avgeriou, and R. Hilliard. Guest editors' introduction: Software architecture: Framing stakeholders' concerns. *IEEE Software*, 27(6):20–24, 2010.
18. Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. The use and effectiveness of user stories in practice. In *Requirements Engineering: Foundation for Software Quality*, volume 9619 of *LNCS*, pages 205–222. Springer, Berlin, 2016.
19. L. Luinenburg, S. Jansen, J. Souer, I. van de Weerd, and S. Brinkkemper. Designing web content management systems using the method association approach. *Proceedings of the 4th*

International Workshop on Model-Driven Web Engineering (MDWE 2008), pages 106–120, 2008.
20. A. W. Meade and S. B. Craig. Identifying careless responses in survey data. *Psychological Methods*, 17(3):437–455, 2012.
21. J. Muskens, R. J. Bril, and M. R. V. Chaudron. Generalizing consistency checking between software views. In *Fifth Working IEEE / IFIP Conference on Software Architecture (WICSA 2005)*, pages 169–180. IEEE Computer Society, 2005.
22. C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Trans. Internet Techn.*, 2(2):151–185, 2002.
23. B. Nuseibeh. To be and not to be: On managing inconsistency in software development. In *Proceedings of the 8th International Workshop on Software Specification and Design*, page 164. IEEE Computer Society, 1996.
24. B. Nuseibeh. Soapbox: Ariane 5: Who dunnit? *IEEE Software*, 14(3):15–16, 1997.
25. B. Nuseibeh, S. M. Easterbrook, and A. Russo. Making inconsistency respectable in software development. *Journal of Systems and Software*, 58(2):171–180, 2001.
26. W. N. Robinson and S. D. Pawlowski. Managing requirements inconsistency with development goal monitors. *IEEE Trans. Software Eng.*, 25(6):816–835, 1999.
27. J. Rosik, J. Buckley, and M. A Babar. Design Requirements for an Architecture Consistency Tool. pages 1–15, 2009.
28. N. Rozanski and E. Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012.
29. R. W. Schwanke and G. E. Kaiser. Living with inconsistency in large systems. In *Proceedings of the International Workshop on Software Version and Configuration Control, January 27-29, 1988, Grassau, Germany*, pages 98–118. Teubner, 1988.
30. G. Spanoudakis and A. Zisman. Inconsistency management in software engineering: Survey and open research issues. *Handbook of software engineering and knowledge engineering*, 1:329–380, 2001.
31. A. Tang, M. A. Babar, I. Gorton, and J. Han. A survey of architecture design rationale. *Journal of Systems and Software*, 79(12):1792–1804, 2006.
32. A. Tang and M. F. Lau. Software architecture review by association. *Journal of Systems and Software*, 88(1):87–101, 2014.

A

Case Descriptions

A.1 Case Study 1 (Expert 1)

Description

The first case study has been executed at a client of the company of Expert 1. A new system must be build for the Union of Security Guards (USG) [1]. A new law states that every security guard in the Netherlands should receive a Securibadge (SB). It is a physical badge that every security guard should wear during shifts, as the government wants more insight and transparency. Initial brainstorming sessions show that the system should consist of two subsystems, indicated in blue. A central system (A) should be developed, a portal which registers a new badge application, and loads the unique ID to a central database in the Netherlands. Every security guard in the Netherlands should be registered at system A. On the other hand, an on-site system B should be built, a check in- and out system, used at every event site, so that guards can register their shifts. There are 70.000 nightly active security guards in the Netherlands, of which 95% starts his shift between 10pm and 11pm. That means that in 1 hour, system B has to deal with 66.500 check-ins. Furthermore, the USG has stated that the system should store the check-in data for at least 5 years. That equals a large amount of data: 66.500 guard ID's, times 5 years, times 220 days in a year, times 2 times for checking in and out. A difficult factor is that most employees work for the large company *Secunite*. It has its **own clock-in systems** at a lot of venues or event sites and a lot of security guards will thus clock-in using the *Secunite* system instead of the system developed for the USAG. Therefore, system B should also be able to accept check-in's from the *Secunite* clock-in system, and match the external clock-in with the respective Securibadge of the security guard.

Executing the CDIM method

It is important to note that not all activities of the CDIM method could be carried out due to time limitations. Two perspectives and four important concerns have been discussed during the instantiation.

[1] The project, domain and company names are fictive due to a non-disclosure agreement.

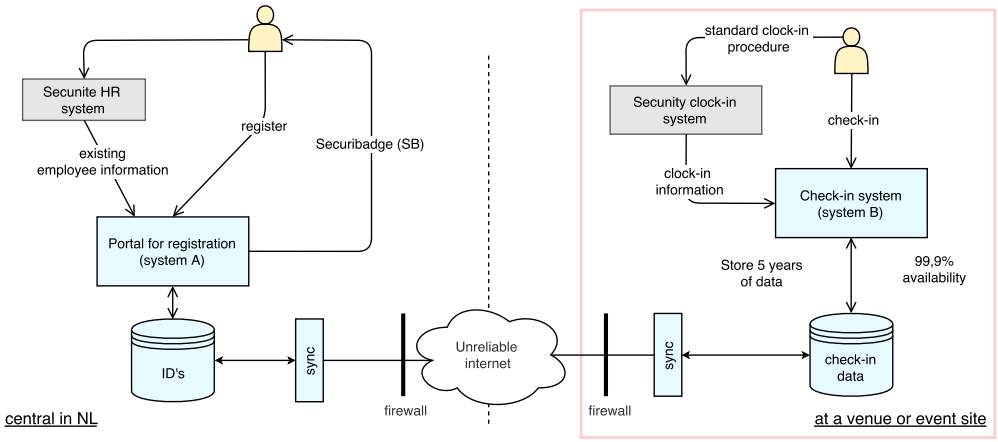


Figure A.1: A contextual overview of the requested systems. Floating text refers to certain constraints, and text on the arrows refers to actions or information.

In the *Instantiation* phase, we set the goal to *discover tradeoffs* since the architecture did not exist yet. A discussion of the situational factors was omitted. The scope was set to include System B, as is indicated by the red square in Figure A.1. Two perspectives were chosen: performance and security. From a security perspective a concern is that the connection established between the Secunite clock-in system and system B should be very secure, because personal information will be transferred from system to system (protect data leakage). Another concern is that system B should be able to discard fraudulent check-ins. From a performance perspective it is important that system B should be able to perform under severe peak load. This is a result by the mandate of USG Netherlands, which states that the system should be available 99.99% of the time. Another concern is that the stability of the connection between system A and B should be high. The design rationale was constructed implicitly by the expert. "*For instance, the concern of preventing data leakage is addressed by encrypting the entire database*". Furthermore, the expert stated:

> "*We address fraudulent check-ins by only accepting incoming check-ins that I trust. By issuing certificates for each system that is connected with system B, the system only accepts certified check-ins.*"

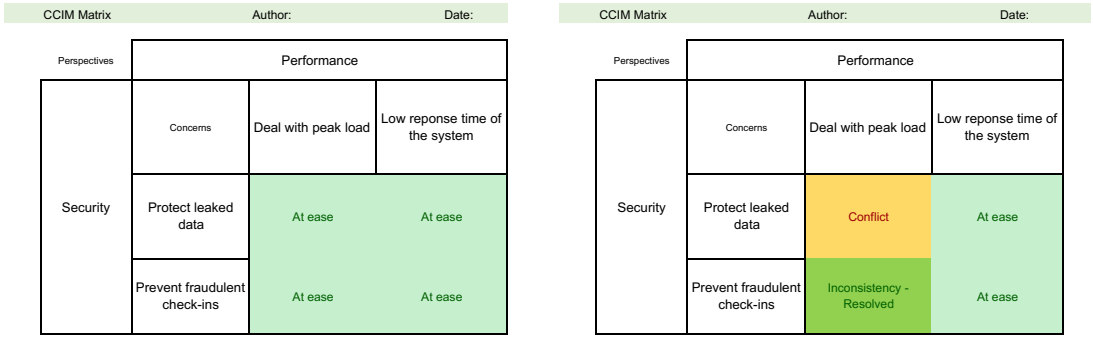
Subsequently, the matrix was constructed (Fig. A.2a): the matrix always starts with all cells 'at ease'. The participant did not go through the activities in *Concern Monitoring* stepwise. He identified overlaps, conflicts and possible solutions to conflicts in one step. The participant identified overlaps in all cells, however only a conflict (and inconsistency) in the two leftmost cells. Due to time constraints, the architect could only discuss a single overlap: 'prevent fraudulent check-ins' and 'deal with peak load' (bottom-left of the matrix).

> "*Certification to prevent fraudulent check-ins has consequences for the performance of the system, as it means that it will be costly and time-consuming to check all the certificates. This concern contradicts the peak load concern. The matrix shows this.*"

Certifying every check-in is inconsistent with the concern that the performance should be maintained during peak load. He stated that the matrix demonstrated this. He solved it by

accepting every check-in during the peak hours, and after the peak hour let the system check whether all certificates are authorized or not. Checking all architectural elements was not possible, since the only design until that time was a whiteboard drawing and assertions and assumptions in the participant's head. The inconsistency was classified using the CDIM *diagnosis* step as having a high impact and a high business value. Engineering effort was not even taken in consideration due to the fact that the other two aspects were so significant. The (possible) inconsistency was solved by letting the system accept every check-in, and afterwards doing an authorization check. Finally, the decision was captured in the matrix, and the impact of the solution was discussed with regards to existing and new concerns. The architect mentioned:

> "*There are other ways to solve it. You can always add more servers to the system, which increase the performance. However, this has likely some influence on the financial picture, you do not want that.*"



(a) Prior to discussion.

(b) With a recorded handling decision.

Figure A.2: The CDIM matrix, before and after discussion.

A.2 Case Study 2 (Expert 2)

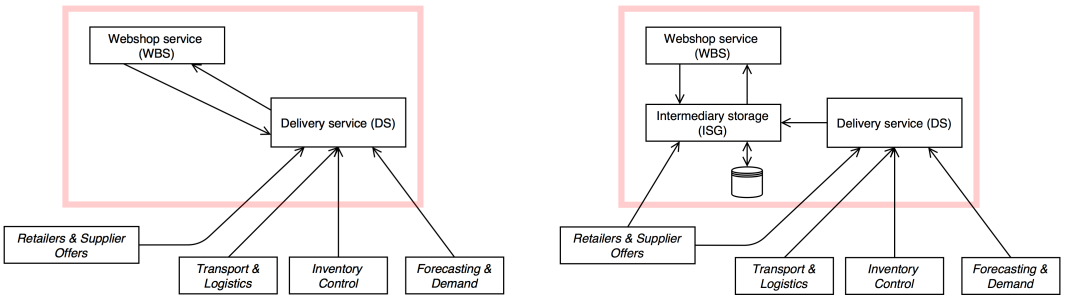
Description

A large webshop in a European country is executing an internal project in which the business wants to improve the estimation and presentation of delivery times of the packages they ship towards customers. Or rather, enable the customers to select their shipping date more precisely. In the current situation, products are classified into multiple product groups, and product groups are assigned a certain shipping date. The shipping date is dependent on the location of the retailer, delivery routes, and other logistic factors. The aim of the project is to enable the webshop to display the delivery date and time for a specific and unique order for a unique customer. In order to do this, the organization must determine how much is in stock exactly, the shipment specifications of the package, which courier is still able to deliver the package, and the unique customer information. Currently, the company is in a design phase of this project; the architects

are brainstorming on how to transform the architecture of the webshop and its supporting systems to a new version. There are already some designs produced, and the four architects are still discussing issues and concerns.

In the current situation, the webshop service (WBS) simply checks to which product group a certain product belongs, in order to find out the estimated delivery time of that product. The estimated delivery times of each product group are hardcoded in the webshop. When a customer adds an item to the shopping cart, the service requests the product group number in order to find out what the estimated delivery time is of a product.

In the proposed situation, the delivery service (DS) will gather the required information, and will be in complete control of the calculation of delivery date and time for all items placed in a customer's shopping cart. The DS will become the heart of the delivery time calculations: "if I have this unique product, when will it arrive at this unique customer?" It retrieves the required information from 'Transport & Logistics', 'Inventory Control', and 'Forecasting & Demand'. Under the circumstances of the new project, every time a product is added to the shopping cart, the WBS has to send a request to the DS. The amount of requests is estimated at 300 per second, during off-peak. Figure A.3a on the left-hand side shows the proposed situation. Figure A.3b is discussed later.



(a) Without intermediary storage service (ISG). (b) With intermediary storage service (ISG).

Figure A.3: A conceptual diagram of the architecture under analysis.

Executing the CDIM method

Due to time limitations not all activities could be carried out. To be consistent with the previous lightweight case study, the number of perspectives and concerns as an input for the CDIM method was limited to two perspectives and four important concerns. In the *Initiation* phase the goal was set to *discovering tradeoffs* because the architecture was not yet constructed. The participant followed a pragmatic approach. The scope was set to contain the webshop service (WBS) and the delivery service (DS). Stakeholders that should be involved were architects and stakeholders from the business. During the next step, two interesting perspectives were identified by the participant. On one hand, the information perspective was identified, with two concerns: real-time timeliness of the delivery time information and correctness of the delivery time information. On the other hand, the concurrency perspective was identified, comprising two concerns: response time and resilience of the system. The next step, *obtain design rationale*, resulted in a set of possible design decisions addressing the identified concerns. To address the concern of real-time delivery

data, the DS gathers and requests information in real-time from the *retailers and supplier offers*, *transport and logistics*, *inventory control*, and *forecasting and demand* systems and services. A design constraint was to maintain a certain hardware specification, and not expand the number of servers for this project. It was observed that while identifying the concerns and the design rationale, the participant already discussed possible conflicts and solutions for identified conflicts:

> "*Doing real-time requests is practically impossible, at least not with the current hardware specifications. A solution to this problem is to introduce a component that will be doing hourly requests to the DLY, and temporary saving delivery dates and times.*"

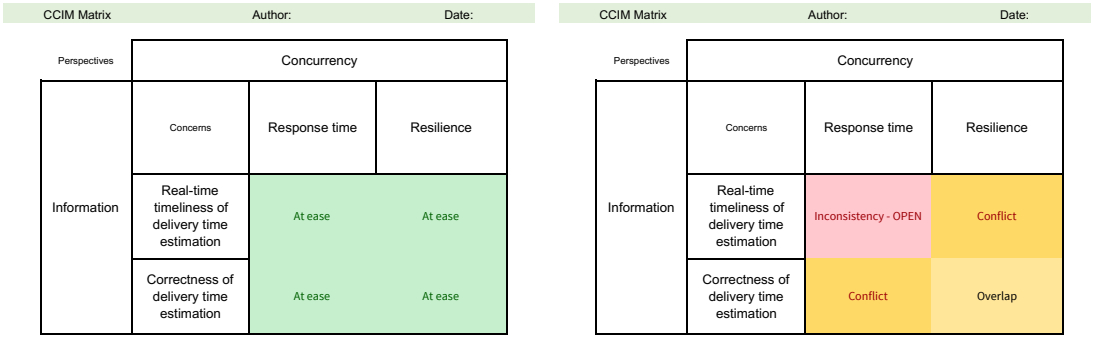
The next activity was *Concern Monitoring*. The matrix was populated as shown in Figure A.4a. It was clear that the participant did not go through all the actions stepwise. Similar to the first case study, the architect discussed overlaps, possible conflicts and inconsistencies, and solving strategies immediately while discussing the matrix. Overlap was found between each set of concerns. In three cells, a conflict was found. Real-time timeliness forms a conflict with the concerns response time and resilience. In addition, the response time is in conflict with the correctness of the information: when the response time of the system is too high, item stocks could be changed and therefore the information is not correct anymore. Due to time limitations, it was decided to focus on the upper left cell of the matrix. Because the participant mentioned he already had several discussions on this topic, he came up with a possible workaround for this possible inconsistency. Their solution is depicted in Fig. A.3b. The following quote summarizes it nicely:

> "*There is a tradeoff at this point; a real-time system would need many additional servers otherwise the response time will be remarkably low. On the other hand, a superb response time is almost impossible if the system should fetch data real-time - with the current hardware specifications. If real-time updates is the most important concern, the system needs a lot more servers. We accept that there is a tradeoff, we search for a solution in between: first, how realtime should the display of delivery times be? We already know that the system is capable of precalculating which delivery dates are possible for a certain product, assuming the product is in stock. Since the only thing that really changes last moment is the stock, we have decided to precalculate possible delivery dates for a product and on the last moment, we do a single request from the WBS to the DS in order to identify whether it is in stock. We simply cannot perform a real-time update of the entire delivery time calculation. However we can do this once an hour, and do stock inventorisation at almost real-time. We accept the inconsistent concerns, they simply cannot be done both, and we came up with a kind of workaround: first step: precalculation of all the factors (not real-time) and than as a second step: real-time stock inspection requests. In case the architecture was built without implementing this solution, we would discover that the DS was receiving way too much load. The inconsistency of two concerns has an outcome that is really critical: a severe load on the DS server.*"

By adding a component (intermediary storage (ISG)) that does hourly requests to the delivery service (DS) and storing important information for products, the WBS is able to present reasonably accurate information on delivery times. During the CDIM cycle, the participant discovered that

the ISG also improved the resilience of the system: when the DS is down, the WBS is still able to use an estimated delivery time. With the new solution, the system is less dependent on a single service (the DS).

The activities *Inconsistency Diagnosis*, *Inconsistency Handling* were done implicitly. The inconsistent concerns were caused by inconsistent demands from the business: real-time information as opposed to a limited budget for hardware setup. The inconsistency was classified as having a high impact if it was built (a severe load on the DS server), and a high business value (important concerns). The inconsistency was eventually solved, by introducing the ISG. Many comments and questions were noted and observed as a result of the CDIM method execution.



(a) Prior to discussion.

(b) With an open inconsistency.

Figure A.4: The CDIM matrix constructed in the second case study, before and after discussion.

B

Design Science Research Cycle

During the analysis of design science research literature we found that the 'three cycle view' of design science research [65] (presented in Fig. 2.1) presents a researcher with an in-depth overview of his research, discussing both the design cycle, the relevance cycle and the rigor cycle (see Sect. 2.2). Hevner stresses the importance of the design cycle [65], however, the 'three cycle view' that he proposes does not depict the activities in a cyclic manner. On the other hand, work of Wieringa [151] makes me believe that the work of Hevner presents a very useful framework, but lacks a certain concreteness, that is needed in a research project (which is the reason this study uses Peffers' DSRM process [120]). Wieringa on the other hand, defines a Regulative Cycle which has four steps: *problem investigation*, *solution design*, *design validation*, and *solution implementation*. Although we see the added value of this cycle, using this for a research project is risky, as it presents the user with little context or guidelines that can be followed during the research. Therefore I propose to combine the work of Wieringa concerning the Regulative Cycle [151] with the 'three cycle' framework of Hevner [65]. It introduces concreteness to the *relevance-* and *rigor cycle* by depicting concrete research steps on the two respective cycles. These research steps are based upon the Regulative Cycle. Figure B.1 presents the Design Science Research Cycle (DSRC). It attempts to illustrate how a design science research project extracts ('scopes') the problem from the *Real World* and establishes *Requirements* that address that problem. Identified *Requirements* can be validated through 'justification' in the *Real World*. The *Requirements* ideally lead to a *Solution* through what is called a 'designing' activity. The *Solution* is ideally 'validated' on the basis of the *Requirements*. Realizing a *Solution* yields an *Implementation*, which should 'verified' on the basis of the intended *Solution* and 'deployed' in the *Real World*. Using 'evaluation', one assesses whether the *Implementation* addressed the problem in the *Real World*. What this DSRC also illustrates, is the importance of knowing that the problem in the *Real World* and the *Implementation* typically lie in the *Environment*, and the *Requirements* and the *Solution* are often part of the *Knowledge Base* of a project. Furthermore, the DSRC presents how the Relevance Cycle dissects the *Real World* and the *Solution*, as executing the Relevance Cycle entails iterating from *Solution* to *Implementation* to *Real World* to *Solution* again. The same goes for the Rigor Cycle, which dissects the *Real World*, *Requirements* and the *Solution*: constantly adding new knowledge from the evaluation in the *Real World* to the analysis, and process the new knowledge in the *Solution*. It is important to note that this research does

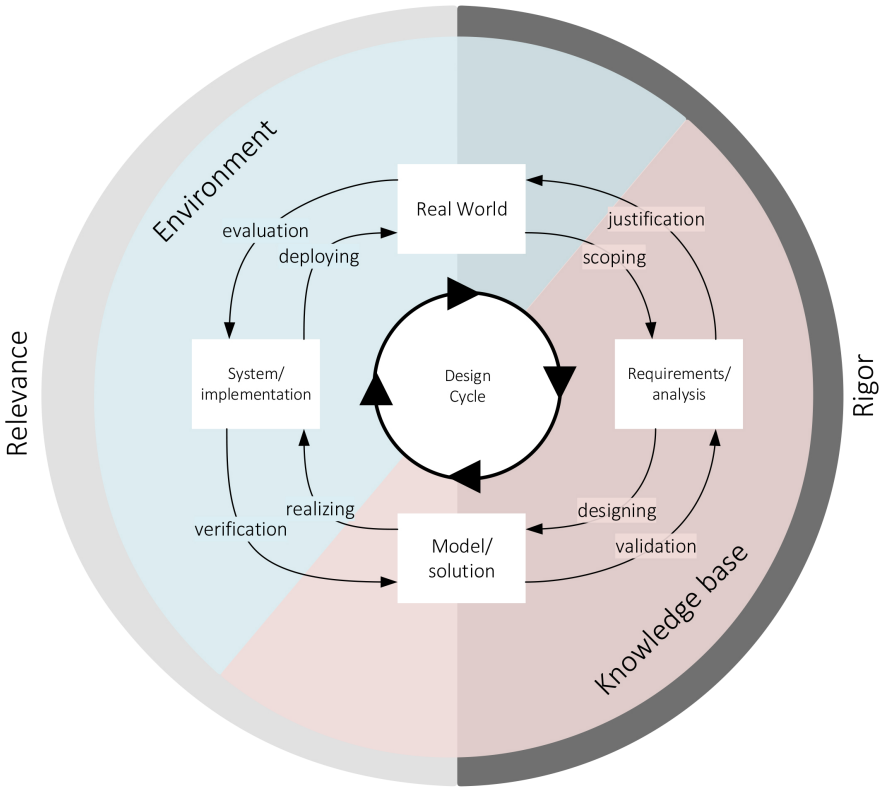


Figure B.1: The Design Science Research Cycle.

not explicitly use or follow the DSRC in this project. The primary reason for this is that the DSRC is not yet evaluated in practice (and evaluation of the DSRC is not within the scope of this research), and can therefore lead to invalid results of the entire project. In Chapter 10 we discuss whether the research activities we have performed resemble the activities of the DSRC, as a basis for further work on this topic.

Revision of the DSCR. Although the Design Science Research Cycle (DSRC) (presented in Appendix B) emerged as a result of analyzing design science research literature, it remains a by-product of this thesis. Therefore, the DSRC is not discussed as a part of this thesis' findings in Chapter 9, but is briefly reviewed here. In hindsight, it is our belief that all activities of the DSRC closely resemble the research activities that we have performed during this study. We aim to justify this by briefly describing how each DSRC activity (or deliverable) coincides with one of the activities in this thesis.

- **Real World**: represented by the problems arising as a result of inconsistencies in software architecture.
  - **Scoping**: requirements were extracted from the experts during expert interviews (Chapter 5).

  - **Justification**: the requirements were justified by literature requirements on correspondence.
- **Requirements**: the requirements have been presented in Section 5.3.
  - **Designing**: the requirements formed an input to the design of the CDIM via the MAA (described in Chapter 6).
  - **Validation**: the CDIM method is evaluated through the use of case studies and expert interviews. The degree to which the CDIM method addressed the stated requirements (validation) is described in Section 9.1.
- **Model**: represented by the CDIM method, the primary deliverable of this research (Chapter 6).
  - **Realizing**: the CDIM method is instantiated through a conceptual prototype in Appendix C.
  - **Verification**: the prototype is not thoroughly evaluated in this study, which provides an avenue for future research. The tool was briefly demonstrated during the expert walkthroughs.
- **Implementation**: represented by the CDIM conceptual prototype, which is carefully constructed, but needs thorough evaluation in practice.
  - **Deploying**: if the CDIM prototype is developed in the future, deploying refers to adoption of the CDIM tool by architects.
  - **Evaluation:** if the CDIM prototype is developed in the future, we can learn from feedback from architects.
- **Rigor Cycle**: the Rigor Cycle in this project consisted of validation of the CDIM method in practice. The Rigor Cycle connects the Real World, the Requirements and the Solution in a cycle that crosscuts the Design Cycle.
- **Relevance Cycle**: the Relevance cycle in this process consisted of using knowledge from evaluation in practice to adapt the CDIM method and the CDIM tool to new versions.
- **Environment**: the organizations that have participated in the research form the environment in which the CDIM method is evaluated.
- **Knowledge Base**: artifacts such as the Method Base Appendix F.1, the association matrix, and the collection of articles, reports and books make up the Knowledge Base.

First of all, there is a close resemblance between the activities of the DSRC and this thesis project. A few activities do not coincide completely: the 'deploying' step is not executed as the DSRC describes, to the extent that the CDIM method has been evaluated by putting it in practice, instead of 'deploying' the conceptual CDIM prototype. We believe that not in all circumstances an *Implementation* has to be 'realized'. For example, when the *Solution* comes in the form of a method, it can be used directly in the *Real World*.

Second, there are cases in which the *Solution* cannot be applied directly in practice. In this case, 'realization' should be addressed. If all the feedback arrows (i.e. the arrows that go counterclockwise) are addressed (and satisfied), theoretically the problem in the *Real World* should be addressed. Third, when one wants to use the DSRC, more attention should be paid to executing the Rigor and Relevance Cycle. Based on the brief review provided above, we carefully suggest that the DSRC could provide a comprehensive framework for guiding future scholarly design science research project.

C Prototype

Various parts of the CDIM method are instantiated in a high-fidelity prototype. This chapter discusses the basic concepts of the prototype tool, and illustrates various screens. The goal of the prototype was to present the participants of the evaluation (in Chapter 8) with a comprehensive overview of how the CDIM method could be used.

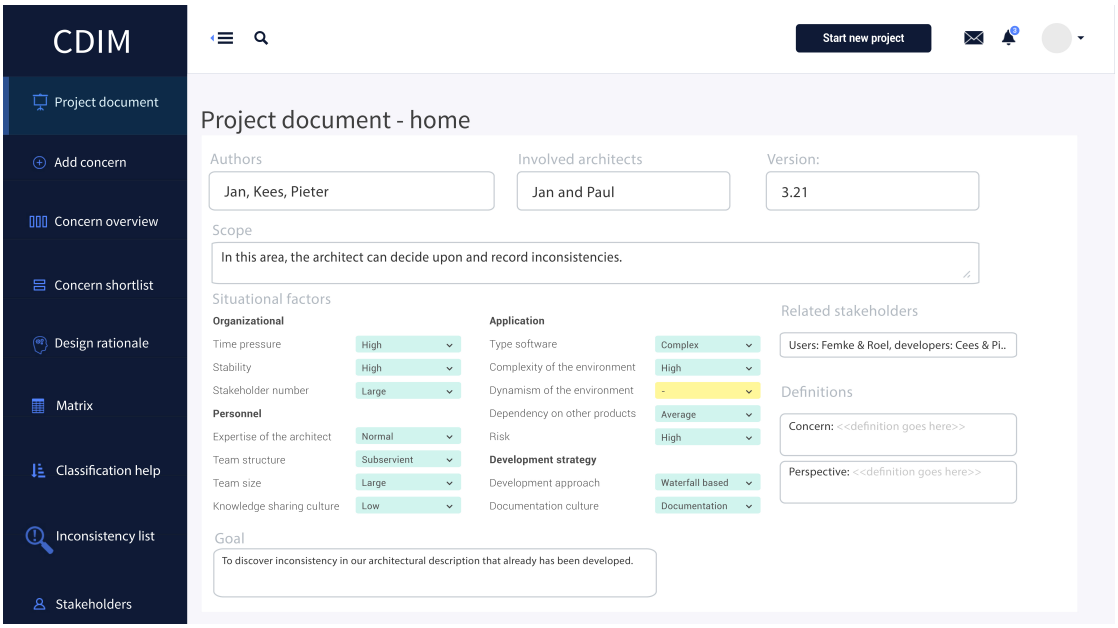


Figure C.1: The landing page of the CDIM tool, at the same time the project overview.

C.1 CDIM Tool

The conceptual tool is a 'horizontal' prototype that is built using InVision[1] and Photoshop. With 'horizontal' is meant that the prototype contains the high-level functionality, but does not contain the lower-level detail of the system [147]. The prototype is a high-fidelity prototype in the form of a clickable demo. We chose to build a high-fidelity prototype since high-fidelity prototypes enable a detailed overview of the functions and flow of an application [131], as was the goal of the walkthrough during the evaluation. With a low-fidelity prototype, we expected that the participants would find it difficult to decide whether they perceived the tool as useful or not.

C.1.1 Overview

The CDIM prototype is a management suite that enables architects to gather and specify concerns, build and maintain a matrix, classify and specify inconsistencies, and record handling decisions. It is important to note that the tool is solely conceptual, it does not have a working interface or responding backend. The prototype is especially beneficial for the systematic architects (the two profiles are discussed in the CDIM method in Chapter 7). The left hand menu bar (see Fig. C.1) contains tabs for almost all of the supported actions within the tool: the home screen (project document), adding concerns, overview and shortlist of concerns, the matrix, classification help, an inconsistency list, and a list of involved stakeholders. Design rationale and 'actions' are currently not supported, as they were not needed for the evaluation. When the architect starts the application, the landing page is a screen (Fig. C.1) containing project specifications, such as the authors (people that work with the tool), the involved architects, the involved stakeholders, which situational profile is adopted, the definitions that are established, and the goal of the CDIM cycle.

[1] InVision is a prototyping tool: `http://www.invisionapp.com/`

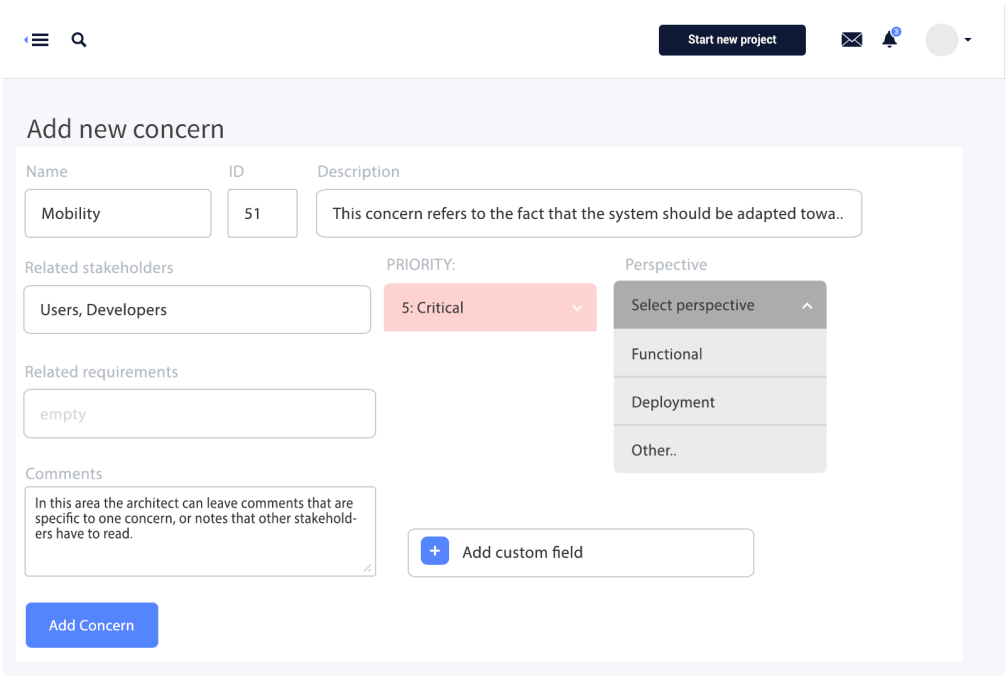


Figure C.2: Adding a new concern in the CDIM tool.

C.1.2 Screens

Adding a concern. After an architect has performed activities in 'Planning', he identifies viewpoints (perspectives) and gathers concerns. Figure C.2 demonstrates the screen in which an architect adds and specifies new concerns. The architect can only fill in the fields that are needed currently. A concern is recorded in terms of its name, a unique identifier, a small though concise description, its related stakeholders, its priority, its associated viewpoint and related requirements.

Concern overview. When a concern is added, it is added to the 'Concern overview', represented in Figure C.3. In this screen, the architect sees an outline of which concerns are gathered, and which are added to the shortlist. As concerns are prioritized and added to the shortlist, they can be used in the matrix. Clicking on a concern brings the architect to a screen in which he can adapt or update information about a concern. From here, the architect can go to the shortlist, and browse through the concerns that are added to the shortlist. The concerns are grouped on the basis of the viewpoint that the architect gave them.

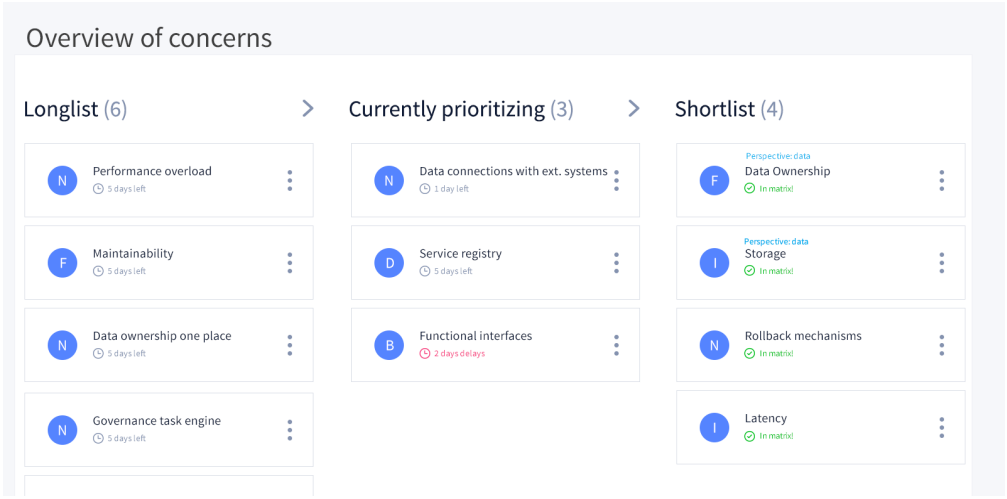


Figure C.3: Overview of concerns incorporated into the tool.

Design rationale. We chose to omit 'design rationale' from the prototype, as the design and documentation of a architecture design rationale differs from organization to organization [141]. Moreover, forcing the architect to use a specific format was discouraged by the experts participating in the expert interviews.

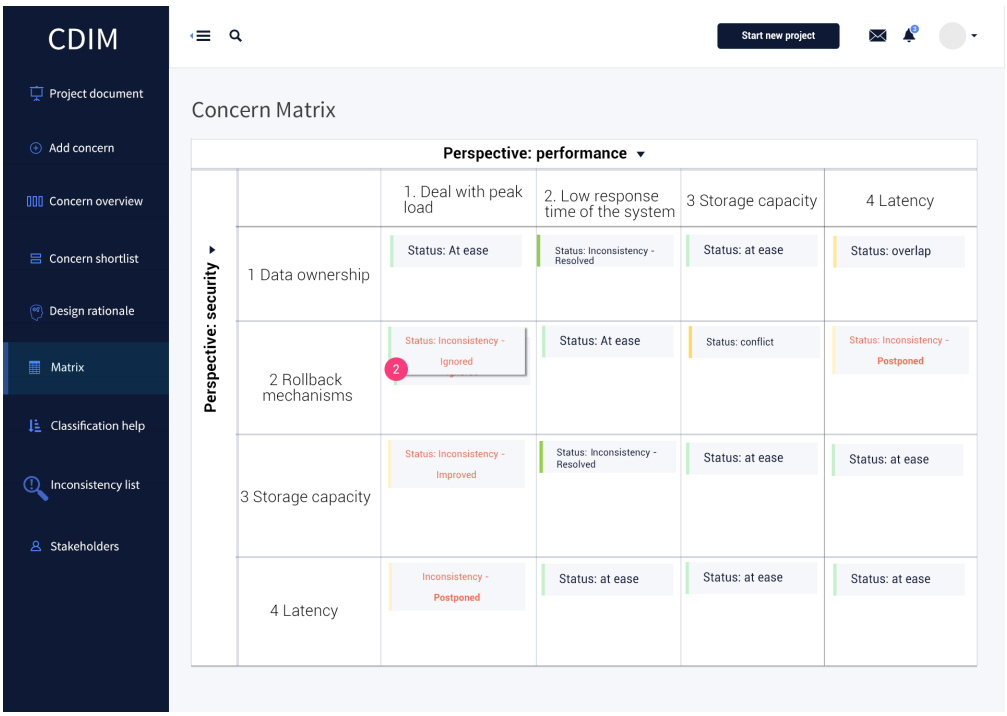


Figure C.4: The CDIM matrix, both axes populated with concerns from two viewpoints. In this case, the same viewpoints have been selected.

Matrix. In the screen depicted in Figure C.4, the architect has decided to build the matrix with the same viewpoint on both axes (both displaying data concerns) in order to evaluate concerns of both viewpoint. When the architect selects a different viewpoint on the horizontal axis, the items change and the matrix' cells are updated with the new viewpoint. The cells are identified by using numbers, making them easily findable. A cell contains a 'status', referring to whether the relation between the two respective concerns yields an overlap, a conflict or whether they are at ease. When an architect has discovered an inconsistency for a particular cell, the status can be changed to 'inconsistency'.

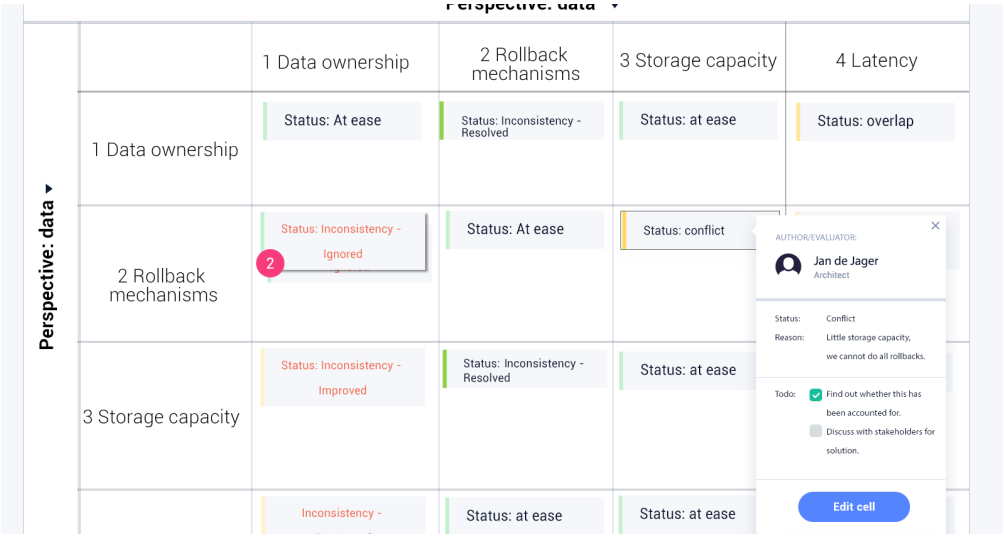


Figure C.5: When an architect hovers over the contents of a cell, a dialog containing extra information appears.

Hovering over a cell with the mouse results in a pop up window, quickly and briefly showing the architect extra information. For example, the architect hovers over the cell that associates 'storage capacity' and 'rollback mechanisms'. The screen depicted in Figure C.5 pops up. The pop up displays information about which architect identified the conflict and what the reason is. The 'reason' refers to a small description of why a particular status was set. 'Todo' refers to tasks or actions that are needed for the cell. The architect can see which to do's have been completed, and which remain to be done.

When an inconsistency has been found, and a handling decision was performed, a cell can be flagged with the corresponding handling action that has been taken. If a cell contains multiple instances[2], the architect clicks on the cell to see a pop up window which presents the number of elements in that cell, vertically aligned Figure C.6. Hovering over one of the instances presents the architect with the extra information (see Figure C.7).

[2] can be overlaps, conflicts or inconsistencies

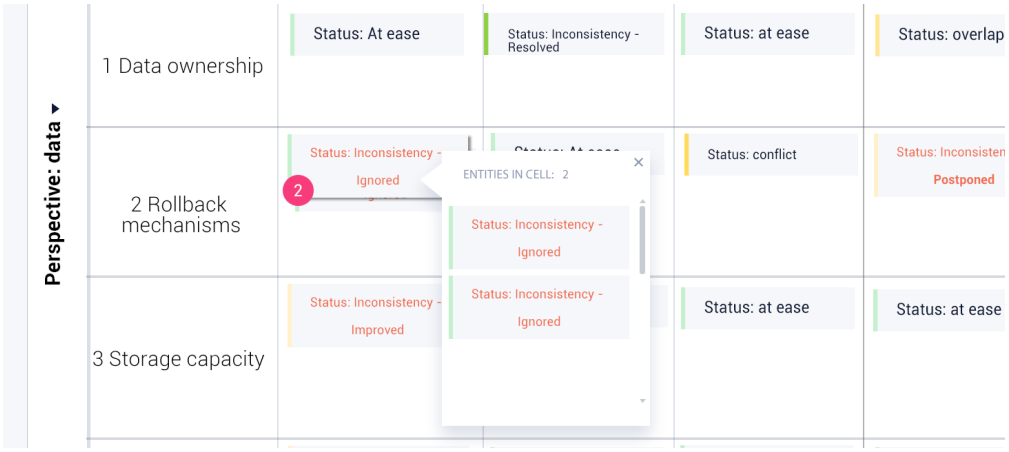


Figure C.6: When a cell contains multiple instances, the architect can check the contents of a cell by clicking on it.

Details of a cell. If the architect clicks on 'edit cell', he zooms in to further details (requirement ER9). The other pop up displayed in Figure C.6 describes a discovered inconsistency in the cell of row 2, column 1. The architect can directly go to details about the inconsistency that has been found, or to the details of the cell. If we return to the example, and the architect clicks on 'edit cell' in the right pop up (row 2, column 3), the screen depicted in Figure C.8 is shown. It represents further details on that cell. In this screen, the architect is able to record and update the status, present a reason for selecting that status, record possible affected elements, and add and update to do's. As long as the status is **not** 'inconsistency', the button right next to the status text box remains gray. In the top right corner, the architect can go back to the matrix. In the matrix, two inconsistencies in cell (2,1) are visible. The architect clicks on the cell two zoom in on its contents and sees the two inconsistencies that have been found. When he hovers over one of the inconsistencies, he sees the important details.

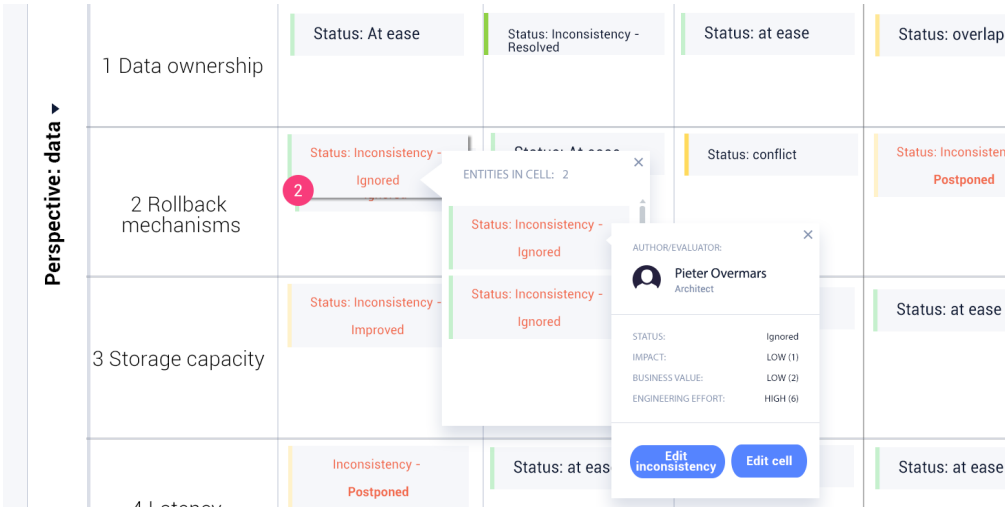


Figure C.7: By hovering over one of the instances, the architect is able to see more detail.

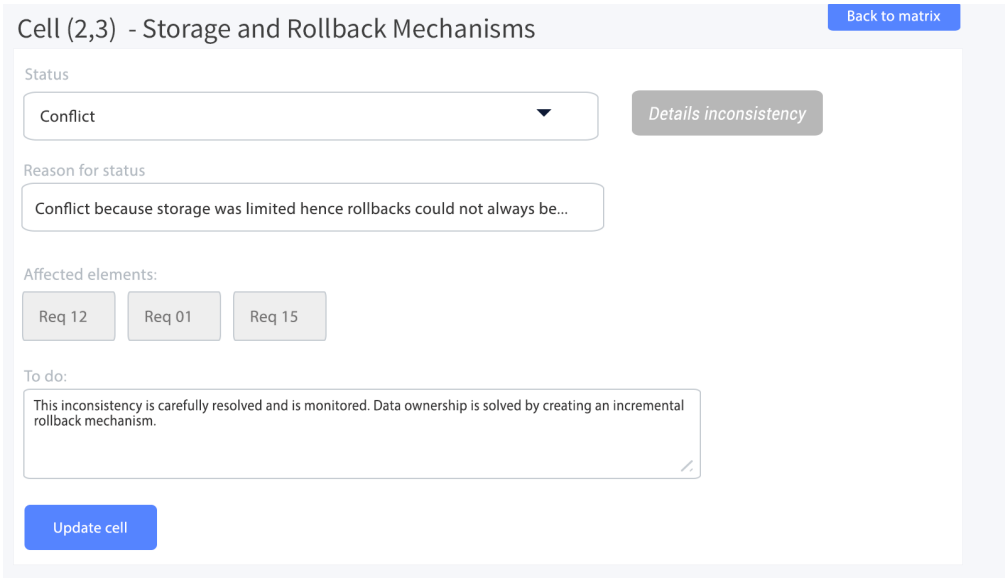


Figure C.8: By clicking on 'Edit cell', the architect is presented details of a cell.

Details of an inconsistency. Clicking on 'Edit inconsistency' brings the architect to the screen presented Figure C.9. This screen enables the architect to record and update details of an inconsistency. Attributes such as the ID and the type can be recorded. In the activity 'Inconsistency Diagnosis' (see Chapter 7) the architect determines the type and the cause of an inconsistency, and subsequently classifies the inconsistency on impact, business value and engineering effort. He can record this classification in the screen as well. The architect is also able to specify a solution and comments for an inconsistency that needs to be solved.

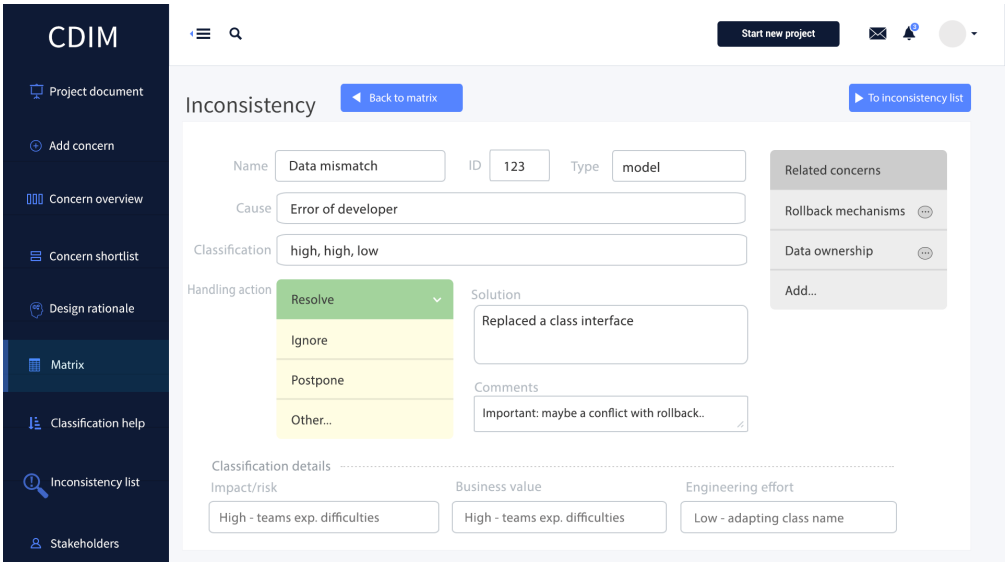


Figure C.9: When 'Edit inconsistency' is clicked, the architect goes to the detailed information about a single inconsistency.

C.2 Summary

The CDIM tool is a conceptual prototype designed to help architects in the process of executing a CDIM method cycle. The tool presents several handles for gathering, recording and communicating concerns. The concerns can be easily loaded in the matrix, and the matrix provides an overview of the overlaps, conflicts and inconsistencies during the architecture process. Altogether, the features in the CDIM tool are expected to help the architect executing the CDIM method. The tool has been demonstrated in the case studies and the expert interviews that have been performed in order to evaluate the CDIM method. They are described in the subsequent chapter.

D

Viewpoints frameworks

Viewpoints Catalogue of Rozanski & Woods [129]. The work of Woods and Rozanski is largely based upon the work of Kruchten [90]. They provide six viewpoints for describing a system's architecture: the functional, information, concurrency, development, deployment and operational viewpoint. Table D.1 gives a brief description of each viewpoint. Figure D.1 depicts the relations between each viewpoint and the system. The structure of the system is described in views conforming to the functional, information and concurrency viewpoints. The implementation constraints are outlined by views that are governed by the development viewpoint. Operational views define the operation and the administration of the system once deployed; views that are governed by the deployment view address deployment related concerns.

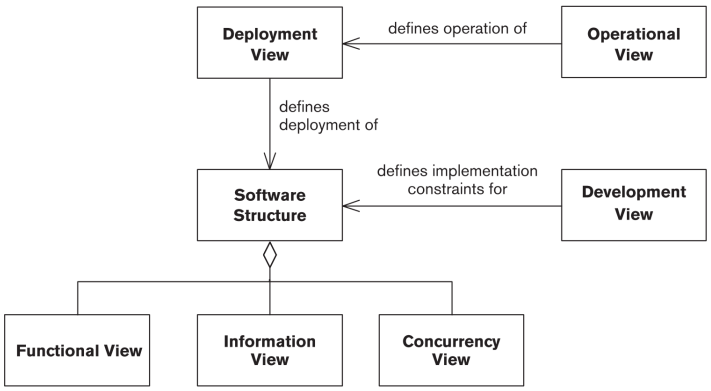


Figure D.1: The different viewpoint classes of Rozanski [129].

Viewpoint	Description
Functional	Describes the system's functional elements, their responsibilities, interfaces, and primary interactions.
Information	Describes the way that the architecture stores, manipulates, manages, and distributes information.
Concurrency	Describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled.
Development	Describes the architecture that supports the software development process. Development views communicate the aspects of the architecture of interest to those stakeholders involved in building, testing, maintaining, and enhancing the system.
Deployment	Describes the environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment.
Operational	Describes how the system will be operated, administered, and supported when it is running in its production environment.

Table D.1: Viewpoint Catalogue from [129].

The 4+1 Views of Kruchten [90]. Kruchten [90] has been very influential by describing four main views that can be used to describe the system's architecture, plus a fifth view that binds the other views together with the use of scenarios. The different views have been described in Table D.2. The fifth view is comprises a small subset of scenarios that demonstrate that the elements of the other views work perfectly together.

View	Description
Logical	The logical view supports the functionality and services the system should provide to end users. Designers decompose the system into a set of object classes that exploit the principles of abstraction, encapsulation, and inheritance [6].
Process	The process view addresses concurrency and distribution, system integrity, and fault tolerance [6]. It also specifies which operations of each class identified in the logical view, are executed by which thread of control.
Development	The development view focuses on the organization of the architecture elements or components in the software development environment. The units of this view are small modules of software, e.g. program libraries or subsystems [90, 129].
Physical	The mapping of the logical, process and development elements on the processing nodes that the system's software will be executed on. This view also represents the system's requirements such as performance and scalability [129].

Table D.2: Kruchten's "4+1 Views" [90].

RM-ODP [126]. The Reference Model for Open Distributed Processing (RM-ODP) is an ISO standard framework for the description of distributed systems [126]. The framework comprises five viewpoints, and was intially created to support the standardization of distributed systems technologies and is reference model for a particular system that is described [153]. So while it was not particularly constructed for pure software architecture purposes, it has found to be sufficient and useful. Table D.3 describes the different viewpoints that can be found in the RM-ODP.

View	Description
Enterprise	Defines the context, the purpose, scope and policies for the system. Furthermore, it addresses the business requirements [144].
Information	Describes the semantics, structure and content of the information required by the system using various schemas [153].
Computational	It describes the functionality provided by the system and its decomposition into functional elements, which interact at interfaces [129, 144].
Engineering	Describes the systems mechanisms and functions that required to implement the desired distribution of the system's objects [144].
Technology	Defines the specific technology that will be used to build the system's functionality and information processing capabilities.

Table D.3: The RM-ODP framework [144].

Siemens Four Views model [69]. Hofmeister and collegues developed a set of four views based upon practices of Siemens. The views provide concrete handles for the creation of views and common issues related to the views. Furthermore, they use UML as a modeling notation, which is widely understood, which makes the views very comprehensible and actionable. However, the views seem a little bit aimed towards developers as stakeholders, as the model lacks a data, operational and a deployment aspect [153].

Viewpoint	Description
Conceptual	The functional structure of the system on a conceptual level.
Module	This view describes the subsystems, modules, interfaces and inter-module dependencies realized in the system, including any form of layering if applicable,.
Execution	This view describes the processes and threads that are responsible for the runtime structure of the system, including the mapping of the modules to the processing nodes.
Code	This view merely describes a development perspective of the system, in the form of source code.

Table D.4: Siemens Four Views model [69]

Views and Beyond [24]. In a book of the Software Engineering Institute called 'Documenting Software Architecture: Views and Beyond', Clements and colleagues [24] aim to provide a comprehensive guide towards documenting software architecture. Their work describes a set of

viewtypes, complemented with a series of architectural styles. The viewtypes are more or less 'specialized' using architectural styles.

Viewtype	**Description**
Module	Views coming from this viewtype describe the elements as modules, which are units of implementation. Modules are assigned units of functionality, which can subsequently be assigned to development teams for implementation. The focus is on code, but not on how code behaves at runtime [29].
Component and Connector (C&C)	In views belonging to the C&C viewtype, the elements refer to components and connectors. Components are principal units of computation, and connectors are the means of communication among components. The focus is less on code, and more on elements such as data stores, parts of the system that run in parallel etcetera.
Allocation	The views coming from the Allocation viewtype address the relationship between code modules and the elements in which the software is executed, for example what kind of processor each software element is executing on.

Table D.5: Views and Beyond [24].

Garland and Anthony's View Set [55]. Garland and Anthony [55] discuss a broad set of views in a practitioner-oriented guide 'Large-Scale Software Architecture: A Practical Guide using UML'. Their set of viewpoints is larger than other viewpoint approaches, resulting in more concrete views on the system; the disadvantage is that fragmentation and consistency play a larger role as a consequence. Table D.6 describes the views that Garland and Anthony define. The table has been adopted from [129].

View	Description
Analysis Focused	Illustrates how the elements of the system work together in response to a functional usage scenario.
Analysis Interaction	Presents the interaction diagram used during problem analysis.
Analysis Overall	Consolidates the contents of the Analysis Focused view into a single model.
Component	Defines the system's architecturally significant components and their connections.
Component Interaction	Illustrates how the components interact in order to make the system work.
Component State	Presents the state model(s) for a component or set of closely related components.
Context	Defines the context within which the system exists, in terms of external actors and their interactions with the system.
Deployment	Shows how software components are mapped to hardware entities in order to be executed.
Layered Subsystem	Illustrates the subsystems to be implemented and the layers in the soft- ware design structure.
Logical Data	Presents the logical view of the architecturally significant data structure.
Physical Data	Presents the physical view of the architecturally significant data structure.
Process	Defines the runtime concurrency structure (operating system processes that the system's components will be packaged into and interprocess communication mechanisms that will allow communication between them).
Process State	Presents the state transition model for the system's processes.
Subsystem Interface Dependency	Defines the dependencies that exist between subsystems and the interfaces of other subsystems.

Table D.6: Garlands and Anthony's set of Viewpoints [55], adopted from work of [129].

E

Method Association Approach

E.1 Full feature list of the CDIM

Table E.1: The full-sized list of features of the CDIM method, as an input for the MAA.

Feature group	**Feature**
Input of the method	Concerns
	Architectural Goals
	Quality Requirements
	Functional Requirements
	Architecture Description
	Business documentation
	Stakeholders
	Architecture
	Design Rationale
Planning	Set goal
	Scope
	Decide on feasibility
	Situational factors
	Prepare & involve stakeholders
Concern Prioritization	Choose perspective
	Gather relevant input
	Prioritize concerns
	Perspective
	Categorization of concerns
	(Short)list of concerns
Concern Management	Wrap up selected concerns
	Obtain design rationale

Continued on next page

Table E.1 – *Continued from previous page*

Feature group	**Feature**
	Establish design rationale
	Populate matrix
Inconsistency Monitoring	Matrix
	Documentation Analysis
	Overlap analysis
	Identify conflict of interest
	Discover tooling possibilities
	Relate to stakeholders
	Check architectural elements
	Discussion of conflicts of interest
Diagnosis	Localization
	Identification
	Classification
	Inconsistency cause
	Inconsistency
Handling	Ignore inconsistency
	Tolerate inconsistency
	Resolve inconsistency
	Postpone inconsistency
	Improve inconsistency
	Bypass inconsistency
	Handling decision
	Request for change
Finalization	Monitor impact
	Add rationale to knowledge base
	Knowledge base
	Change report
	Work proposal
	Present results

E.2 Association Matrix

Feature groups	Features		Methods	SAAM					ATAM					APTIA						ALMA				NUSEIBEH			SAAMER			SBAR	TARA					
			Method Fragments	Architectural Input	Scenario development	Architecture description	Scenario interaction	Overall evaluation	Presentation	Analyze approaches	Brainstorm and prioritize	Analyze scenarios	Present results	Gather input	Determine focus	Analyze architecture	Design approaches	Rank the alternatives	Make decisions	Goal Selection	Describe software architecture	Scenario Elicitation	Scenario Evaluation	Diagnosing inconsistency	Handling inconsistency	Monitor consequences	Gathering information	Modeling usable artifacts	Evaluating	Define set of scenarios	Context Diagram and Requirements	Functional & Deployment views	Code Analysis	Requirements Assessment	Identify & report findings	Deliver findings and recommendations
				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Features	Input of the method	Concerns	1											×																						
		Architectural Goals	2	×					×					×													×										
		Quality Requirements	3	×					×					×													×				×						
		Functional Requirements	4	×					×					×													×				×						
		Architecture Description	5	×					×					×													×										
		Business documentation	6						×					×																							
		Stakeholders	7		×				×	×			×	×								×					×										
		Architecture	8	×					×							×					×						×					×					
		Design Rationale	9														×	×																			
	Planning	Set goal	10																	×										×						
		Scope	11												×																						
		Decide on feasibility	12																	×																	
		Situational factors	13																	×												×					
		Prepare & involve stakeholders	14		×				×	×			×	×								×					×	×									
	Concern Prioritization	Choose perspective	15																								×										
		Gather relevant input	16											×																	×	×					
		Prioritize concerns	17					×			×							×										×									
		Perspective	18																																		
		Categorization of concerns	19																									×									
		(Short)list of concerns	20							×								×																			
	Concern Mgmt	Wrap up selected concerns	22											×																					×		
		Obtain design rationale	23														×	×	×																		
		Establish design rationale	24														×	×																			
		Populate matrix	25																									×									
	Inconsist. Monitoring	Matrix	26																									×									
		Documentation Analysis	27			×										×											×										
		Overlap analysis	28				×																						×								
		Identify conflict of interest	29				×												×										×								
		Discover tooling possibilities	30																													×					
		Relate to stakeholders	31		×				×	×			×	×													×		×								×
		Check architectural elements	32			×										×																×					
		Discussion of conflicts of interest	33								×							×	×										×								
	Diagnosis	Localization	34																					×													
		Identification	35																					×													
		Classification	36									×		×										×									×				
		Inconsistency cause	37																																		
		Inconsistency	38																					×	×	×											
	Handling	Ignore inconsistency	39																						×												
		Tolerate inconsistency	40																						×												
		Resolve inconsistency	41																						×												
		Postpone inconsistency	42																						×												
		Improve inconsistency	43																						×												
		Bypass inconsistency	44																						×												
		Handling decision	45																×																		
		Request for change	46																																		
	Finalization	Monitor impact	47																×				×			×											
		Add rationale to knowledge base	48																×																		
		Knowledge base	49																×																		
		Change report	50																×																		
		Work proposal	51																							×											
		Present results	52										×																							×	×

Figure E.1: The association matrix used in the approach.

F

Method Base

F.1 PDDs of the candidate methods

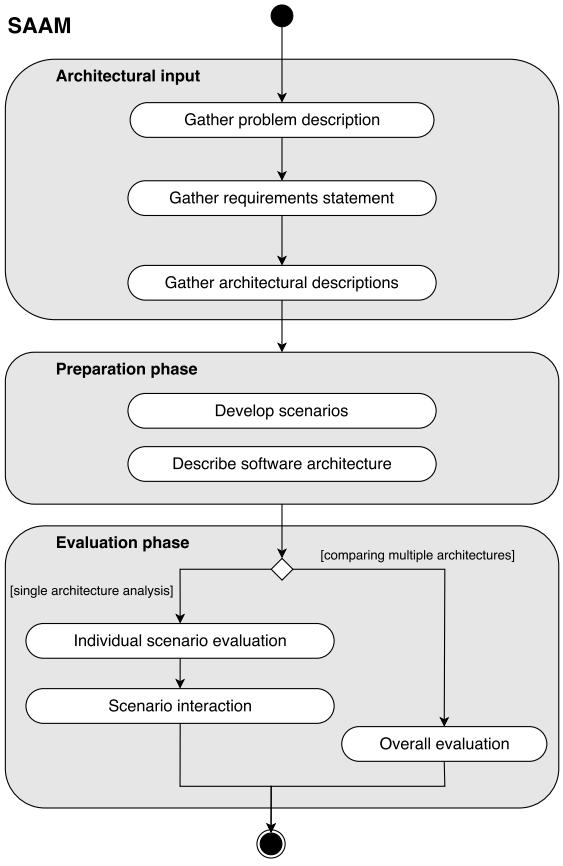


Figure F.1: PDD of SAAM.

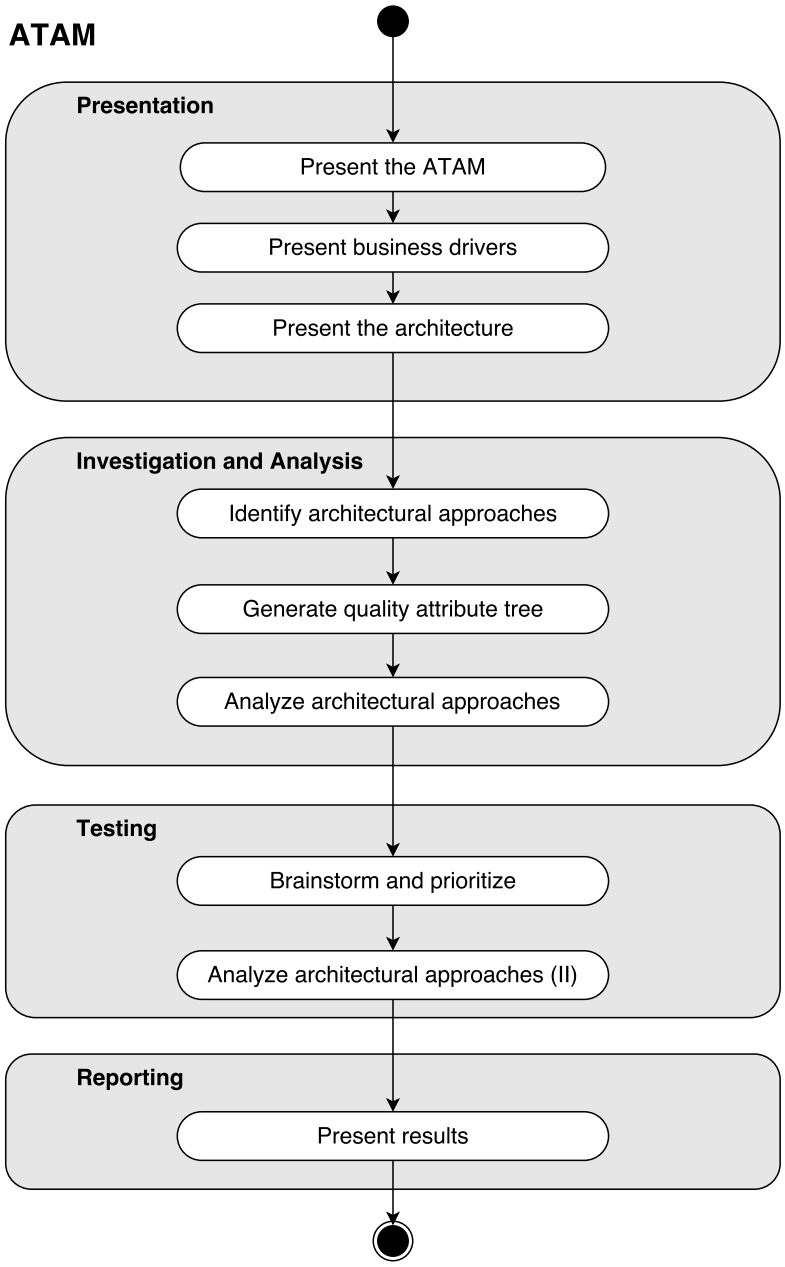


Figure F.2: PDD of ATAM.

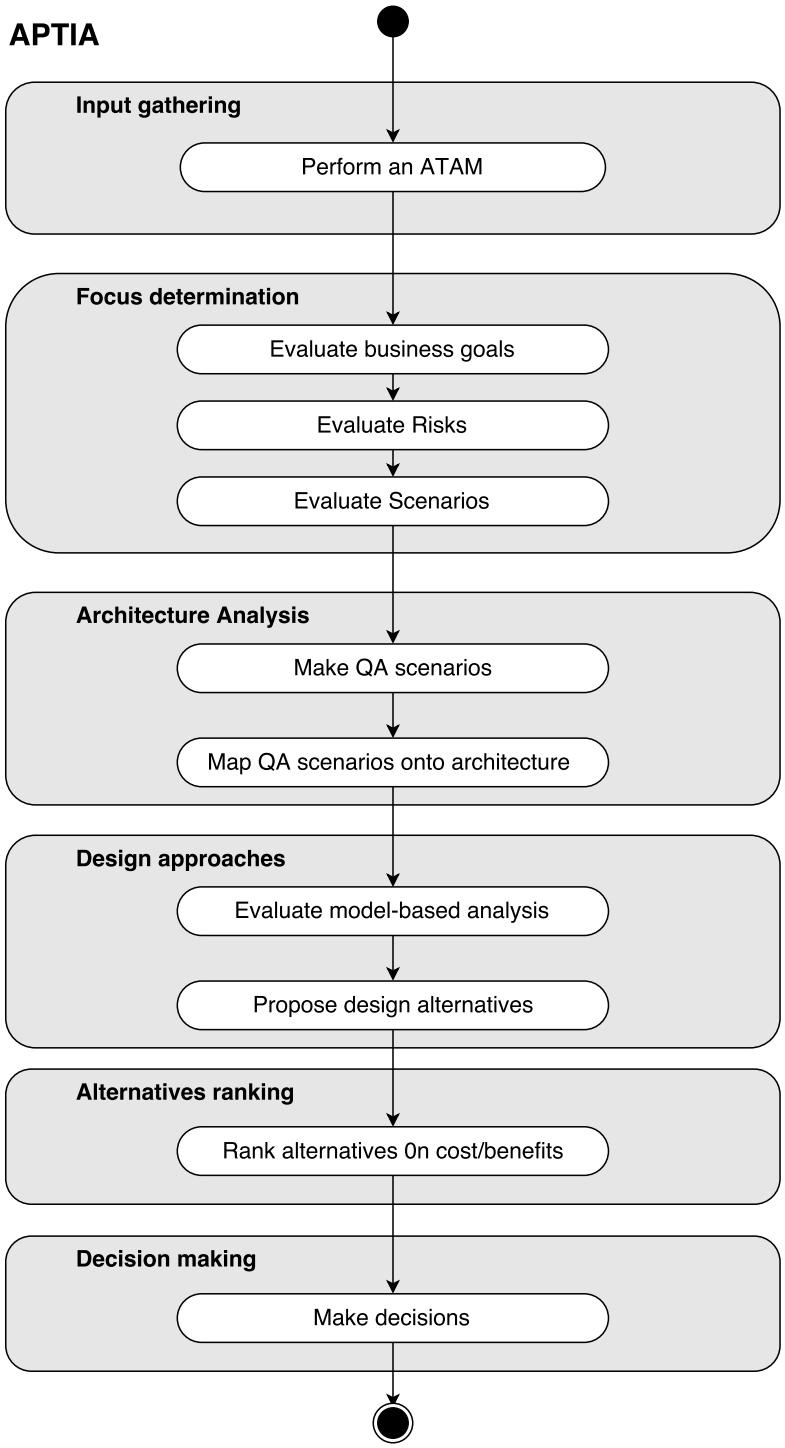


Figure F.3: PDD of APTIA.

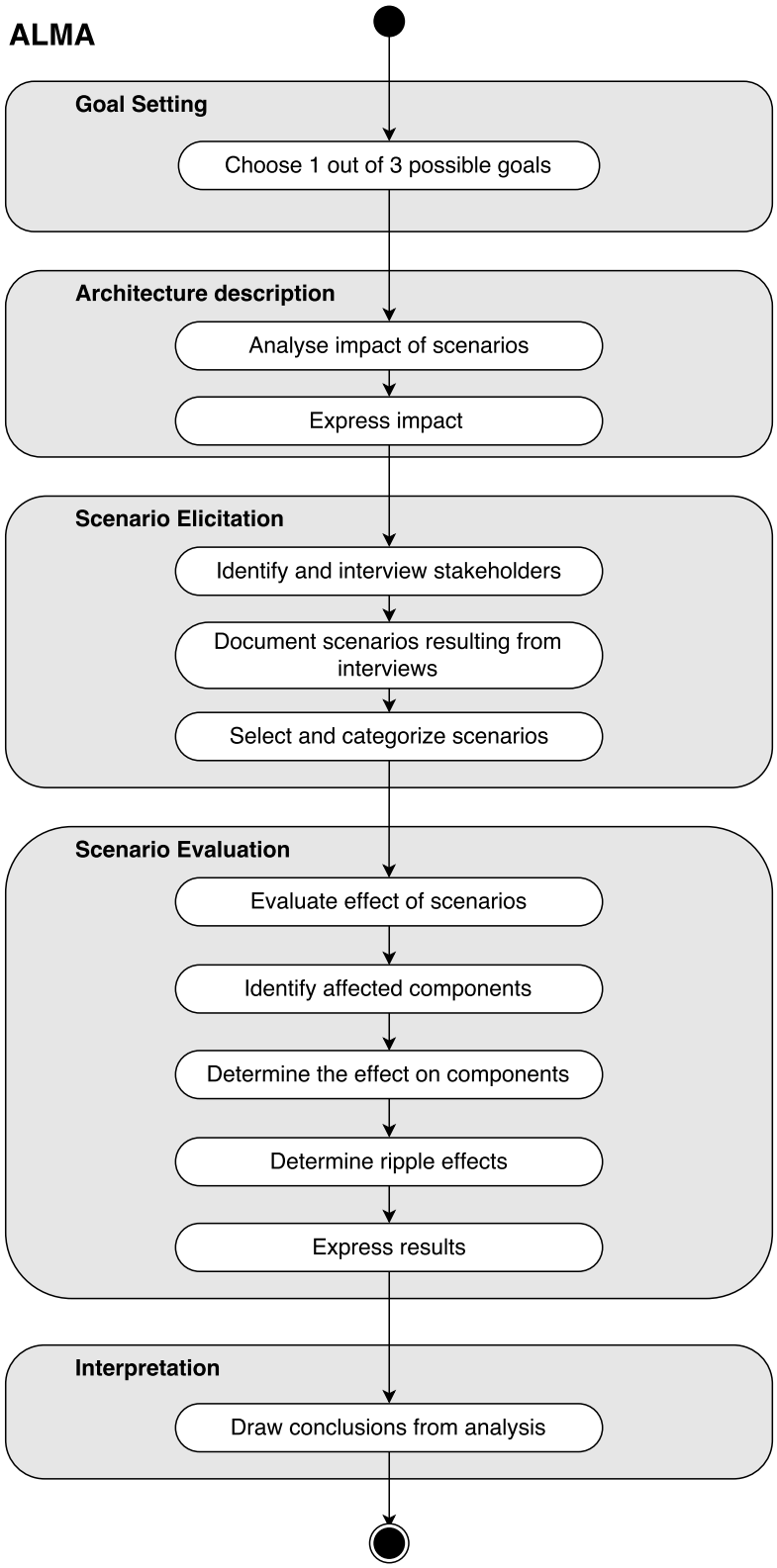


Figure F.4: PDD of ALMA.

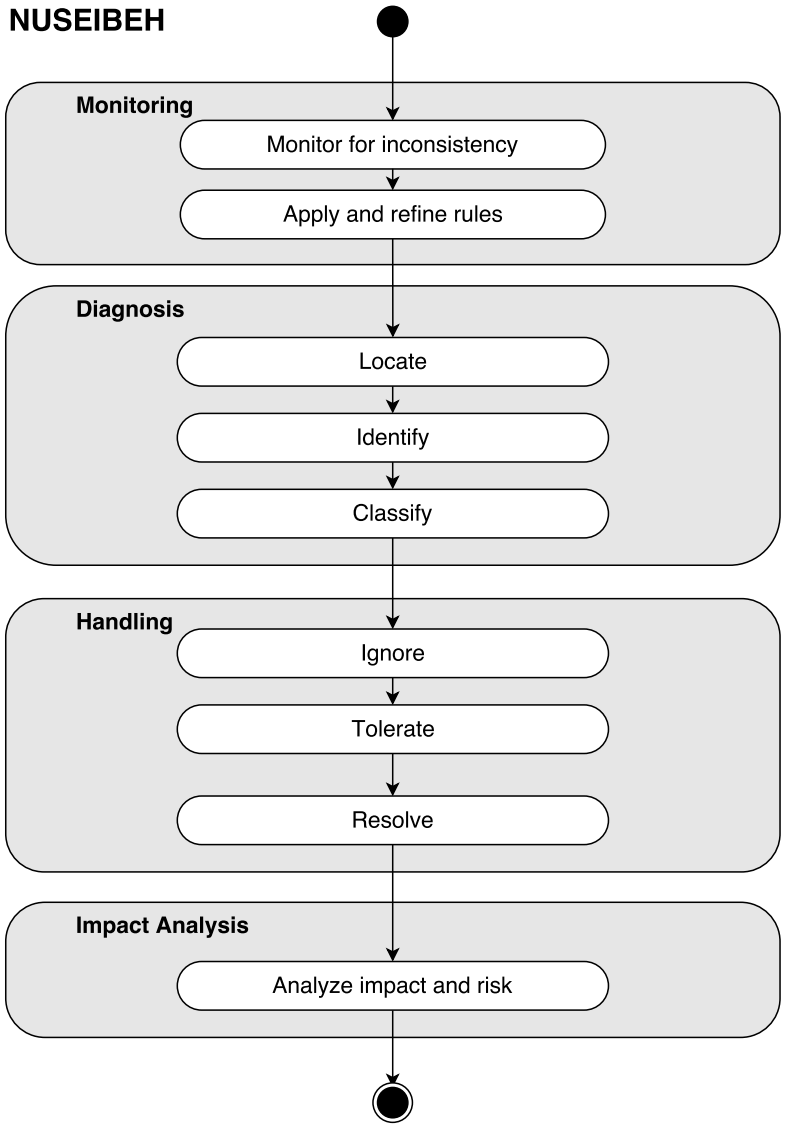


Figure F.5: PDD of the framework of Nuseibeh [116].

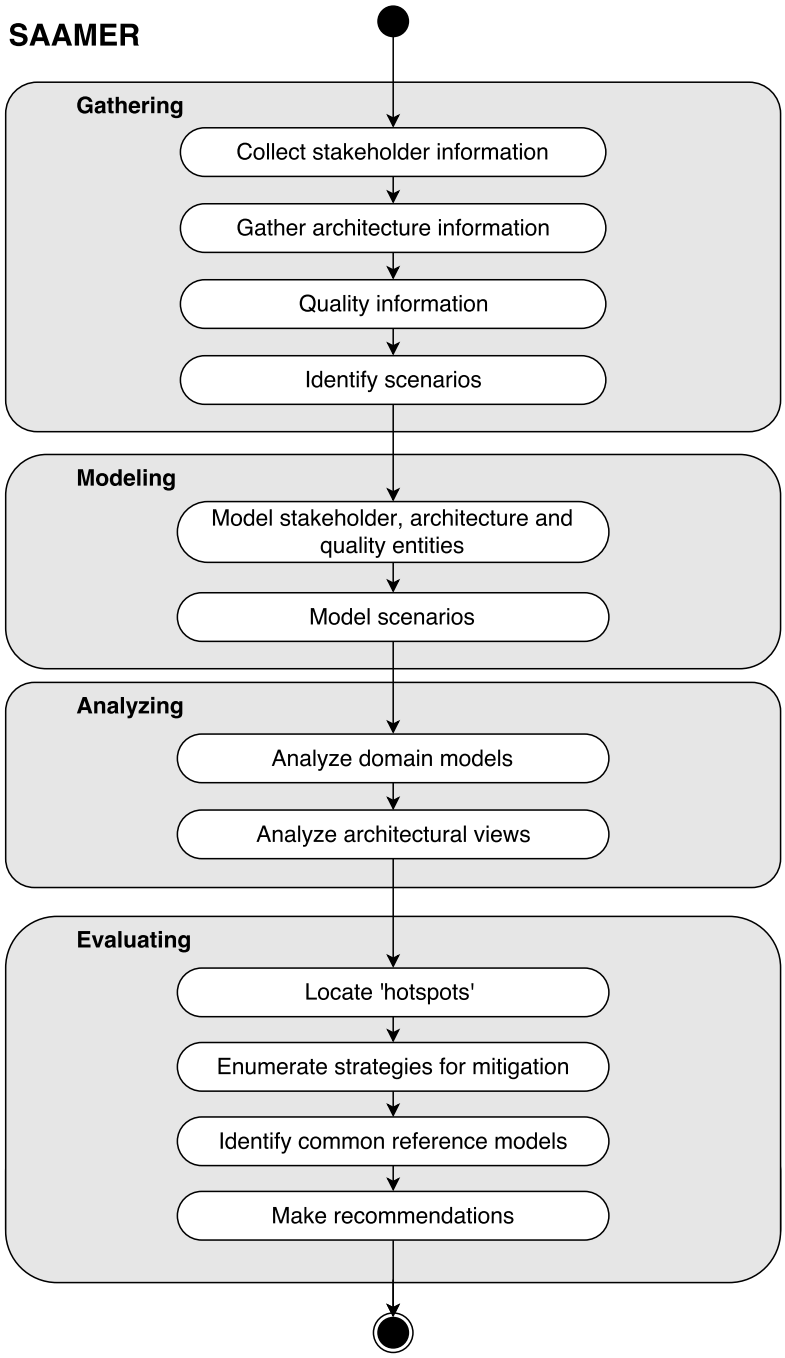


Figure F.6: PDD of SAAMER.

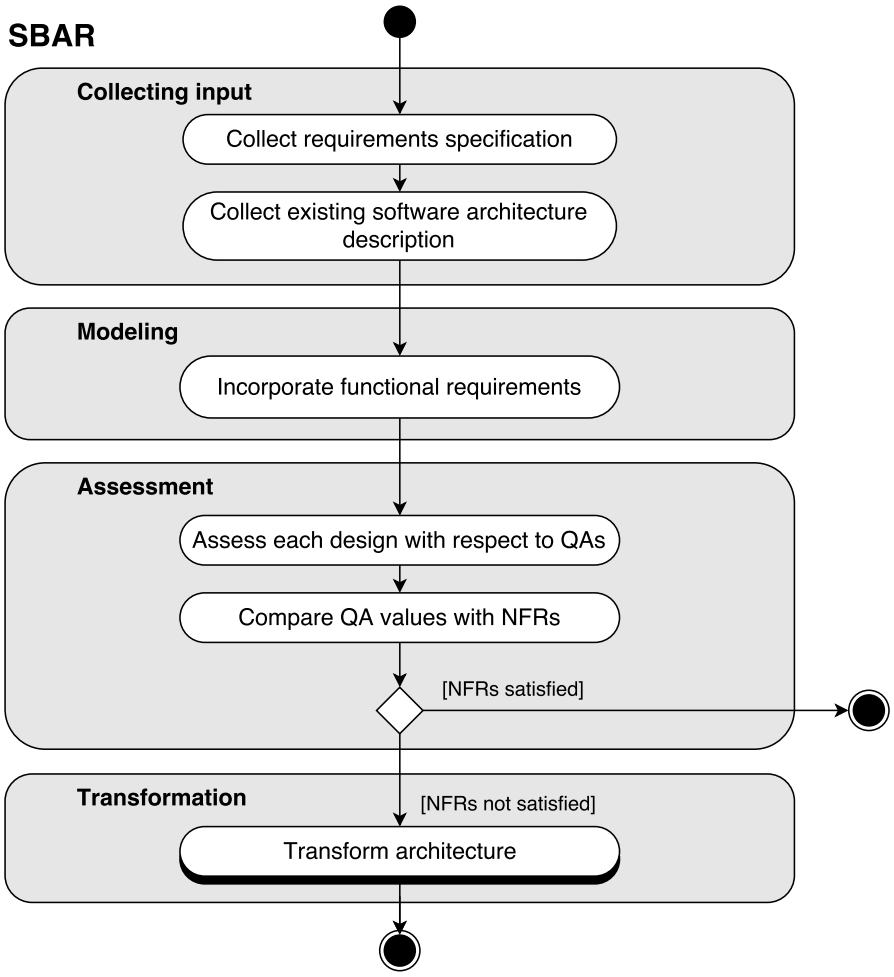


Figure F.7: PDD of SBAR.

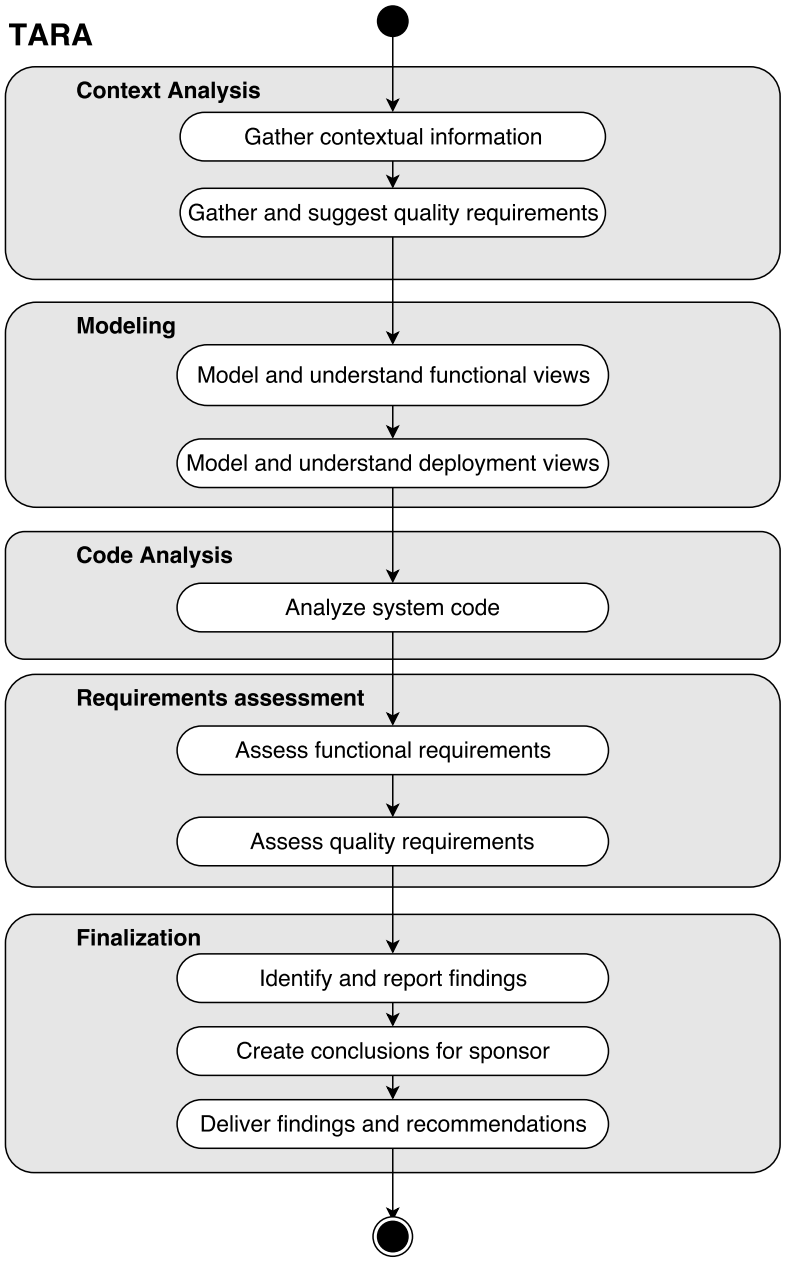


Figure F.8: PDD of TARA.

F.2 PDDs of the CDIM method

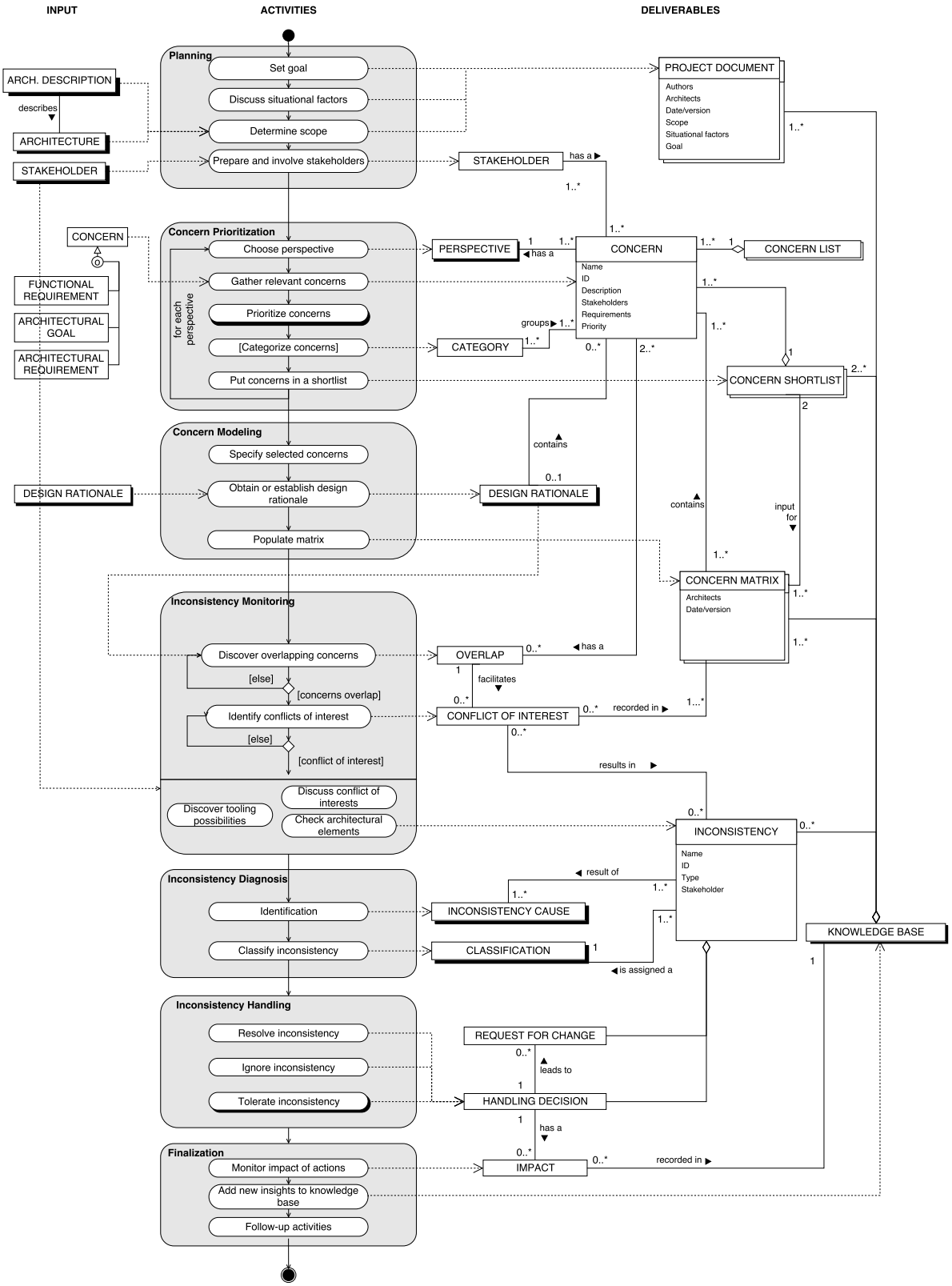


Figure F.9: A process-deliverable diagram of the first version of the CDIM method.

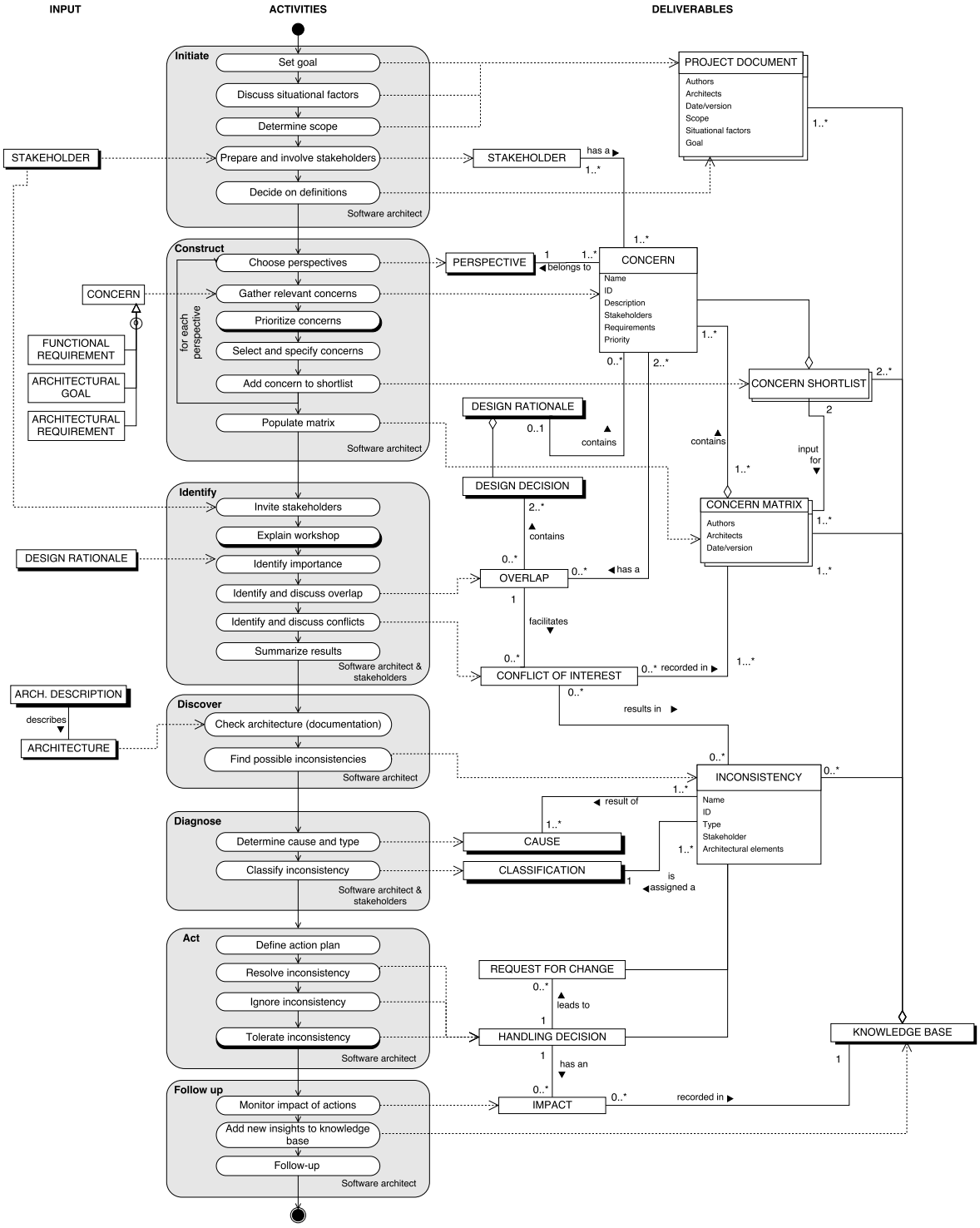


Figure F.10: A process-deliverable diagram of the **final** version of the CDIM method.

G

Excel Template that goes with the method

CCIM Project Document concerns >> **Date:** 28-02-16

Authors:	Diederik	Rene	Jan	
Involved architects:	Diederik	Paul	Rene	Jan
Version:	1.02			

Scope:

The scope has been set to the functional area of the *delivery service WEAP and the the connection with the database.*

Situational approach:

Organizational		**Application**	
Time pressure	high	Type software	complex
Stability	high	Complexity of environment	high
Stakeholder number	large	Dynamism of environment	-
Personnel		Dependency other products	average
Expertise of the architect	normal	Risk	high
Team structure	subservient	**Development strategy**	
Team size	large	Development approach	waterfall based
Knowledge sharing culture	low	Documentation culture	documentation

Result: systematic profile

Stakeholders:

Developers: Cees & Pieter; users: Femke and Roel; manager: Artze

Definitions:

Concern:	high-level aspiration and/or requirement that one of the identified stakeholder groups has for the architecture. Can also be specified rules or regulations from our domain.
Perspective:	one of the 4+1 viewpoints we use regularly: logical, physical, process, development

Goal:

To discover inconsistency in our architectural description that already has been developed.

Figure G.1: A complete project document used in Excel.

H Transcripts

H.1 Development: Expert interviews

Expert 1, EXPERT INTERVIEW, 15-1-2016, 11:00-12:00

1. Wat is je rol binnen Company X?

Ik ben principe IT architect. Ik ga benoemd worden in de rol van practice lead software engineering. In projecten heb ik vaak de rol van SA. Ik verzin de oplossing en help deze dan tot een goed eind te komen, ik zit vaak in de reviewende modus op het moment dat iemand anders een architectuur heeft neergezet. Als klanten een uitdaging hebben dan kom ik heel vaak adviseren.

12:00

2. Hoe worden inconsistenties bij jullie gecheckt?

Ze zijn er zeker weten, dat gebeurt altijd. Wat je vaak ziet is dat de projecten waar je mee bezig bent, zijn zodanig groot en zodanig complex dat je het niet allemaal kunt overzien. Wat je dan gaat doen is dat je gaat modelleren en abstracties maken. En je ziet dingen over het hoofd. Het is inderdaad lastig om al die modellen met elkaar in lijn te brengen. Ik ga niet bewust om met inconsistenties, ik heb geen werkwijze of methode om inconsistenties te vinden en te managen. Het is vaak iets waar je tegenaan loopt, en op basis van intuitie ga je er dan mee om.

Wat ik wel doe in de praktijk, is een hele sterke focus leggen op waar het nou echt om draait in de architectuur. Architectuur draait om de beslissingen die lastig terug te draaien zijn, en 2) de beslissingen waarvan de consequenties kostbaar zijn. Dus waar je je op richt is onmiddelijk toegaan naar de risico's en de cost drivers. En als het dan gaat om inconsistenties, als je dan een inconsistentie hebt in je architectuur die weinig risicovol is of waarvan de kosten laag zijn om het op te vangen. Dat zien we dan wel, dat gaat goedkomen.

Op het moment dat er een inconsistentie is waarbij je hier een beslissing neemt op basis van een aanname terwijl er hier een andere aanname is die lastig terug is te draaien, en we zijn al bezig. Als die inconsistentie dan aan het licht komt als we al bezig zijn, dan heb je een serieus probleem.

Een groot registratiesysteem in de verzekeringswereld, voor het vastleggen van claims. Het was een platform waarop diverse verzekeraars moesten kunnen aansluiten. Een enorm sterke focus was daar op het snel kunnen opleveren, het grootste belang was dat het systeem er binnen een jaar moest komen te staan. Er was een ander belang, dat je altijd terug kon gaan in de tijd; dat alles herleidbaar moest zijn. Het probleem was van origine nog niet geautomatiseerd, dus je kwam tijdens het modelleren van het probleem tot redelijke complexe datamodellen. En als je daar het aspect bij pakt dat alles herleidbaar moest zijn qua tijd, dan explodeert dat datamodel van complexiteit.

17:00

Ze hebben toen het belang om binnen een jaar live te kunnen gaan, en daar is men op gaan ontwerpen. Het belang van dat tijdsaspect heeft men schijnbaar bewust genegeerd. Toen ze wilden opleveren, werd het niet geaccepteerd door de klant. De klant vond het belangrijker om dat tijdsaspect erin verwerkt te zien, dan dat het binnen een jaar live moest. Echter, het systeem stond al, en toen ze dat versie/tijdsaspect erin wilden krijgen, dat was niet te doen. Daar moesten we uiteindelijk zeggen dat we het hele project moesten stoppen.

Die afweging is heel belangrijk; het ene belang maakt het zo complex, dat kunnen we niet waarmaken, en dat gaat geheid fout. Tenzij we met elkaar in gesprek gaan.

> Vraag: bij Company X wordt dus niet gebruik gemaakt van een bepaalde methode of techniek om inconsistenties inzichtelijk te maken?

Nee, maar we hebben wel een strakke methode om gedurende het hele project risico en kosten inzichtelijk te hebben en dat als basis te gebruiken voor de architectuur. Heet RCDA (Risk Cost Driven Architecture).

> Vraag: Als je een inconsistentie vindt, hoe los je dat dan op? Hoe maak je de keuze om er een bepaalde actie te ondernemen?

Wat je 9 van de 10 keer hebt, als je tegenstrijdige belangen hebt, is dat je ook verschilende stakeholders te pakken hebt. *Het is heel belangrijk om die belangen te associeren met de stakeholders* die je hebt. Wat doe ik om een keuze te maken? Die stakeholders bij elkaar brengen: met ze praten, het uitleggen. De enige manier is om de stakeholders bij elkaar in een ruimte zet. In de praktijk doe ik dat ook.

> Heb je daar een voorbeeld van? 21:30 min

Bij X speelt dat. Daar is een architectuur gekozen, en in die architectuur hebben ze bepaalde principes doorgevoerd. Er moet nu iets functioneels bijgebouwd worden: werkzoekende hebben een cv, en werkgevers hebben een vacature. Werkzoekende zoeken naar vacatures, en werkgevers zoeken naar cv's. Werkgevers als werknemers kunnen hun zoekopdracht opslaan zodat ze hem later nog een keer uit kunnen voeren. Nu wil de werkgever notities kunnen maken bij vacatures en een cv, of bij de combinatie van zoekopdracht en cv. Door de gekozen architectuur, waarbij de zoekopdrachten helemaal van vacatures zijn gescheiden is dit heel erg complex. Hele kostbare verandering: voor iets ogenschijnlijk simpels.

Dan heb je dus 2 belangen: de business wil iets ogenschijnlijks functioneel simpels, maar heeft daar maar een bepaalde hoeveelheid geld voor over. Het andere belang is dat we wel een bepaalde architectuur hebben, waarin 2 onafhankelijke componenten met elkaar moeten communiceren. De afweging is dan, dat je kan vasthouden aan de dure oplossing; of ik hou niet vast aan die architectuur, en dan heb ik een goedkope oplossing. Alleen ja, dan gaat mn onderhoudbaarheid er weer aan.

En dan gaan we met de ontwikkelaars zitten, die zijn natuurlijk ook verantwoordelijk voor de onderhoudbaarheid van het systeem. Dus hun belang is het systeem ook conform de architectuur en zou onderhoudbaar mogelijk op te leveren (ook mijn belang als architect). En je gaat ook met de business zitten; wat zijn de ommissies die we kunnen doen? In dit geval waren beide kampen heel star. In dit geval bleek dat het gat te groot was.

> 26:00
> *Het belangrijkste aan architectuur is misschien wel de beslissingen die lastig terug zijn te draaien, om die goed in kaart te brengen.*

Waar het overigens ook 9 van de 10 keer wringt, en dat bedenk ik me nu, is je technische oplossing en gebruiksvriendelijkheid. Vanuit de business wil men altijd hele gebruiksvriendelijke dingen, (quality requirement), en om die dingen gebruiksvriendelijk te maken, loop je tegen technische onmogelijkheden aan. Het kan gewoon niet, tenzij we er heel veel tijd en een hele dure oplossing aan besteden, en in het diepste van een framework dingen aan te passen. Nu zijn we gewend om drag-and-drop interfaces te hebben, maar 15 jaar geleden nog niet. Maar vanuit gebruiksvriendelijkheid aspect, kunnen we dit niet gewoon zo en zo doen, ipv al die tussenschermpjes en zo. Tja, ik zat daar dan met een web framework, en dat kan gewoon niet.

> 28:30

> Zijn er op dit moment dingen waar je van zegt, dat zou ik in de praktijk anders willen?

Ik ben een enorme fan van 4+1, waarom ben ik daar enorm fan van? Het is niet zo zeer dat ik er super in geloof, maar meer dat het een heel helder template is, wat mij dwingt om op bepaalde manieren naar software te kijken. Gedwongen om vanuiĳt een use case perspectief te kijken, ik word gedwongen om uit een deployment view te kijken etc. Het is een rete simple template. In no time heb ik een architectuur beschreven, op een niveau dat het werkbaar is en dat we verder kunnen. **Dus als ik een instrument heb, wat mij dwingt om naar bepaalde dingen te kijken, en wat simpel in het gebruik is, daar ben ik naar op zoek.** Die inconsistenties in een architectuur zijn interessant op het moment dat ze de risicovolle en dure aspecten raken, als ik een simpel instrument heb om die belangen vanuit dat perspectief snel te kunnen onderzoeken, dat is waardevol. Want dan kan je het namelijk onmiddelijk in je praktijk toepassen.

> In hoeverre leggen jullie in een rationale ontwerpbeslissingen vast?

Wat ik altijd wel doe in mn architectuurbeschrijvingen, in die 4+1 word ik gedwongen om naar een use case perspectief te kijken, maar je word ook gedwongen om naar je architectural concerns te kijken, die in ieder geval te inventariseren. Ook je andere perspectieven, daar werk je dingen uit, en wat ik wel altijd probeer is te refereren naar die architectural concerns.

> Dus je legt voor jezelf een soort van rationale aan. Correct.

Mijn concerns nummer ik altijd keurig. En dan in de teksten verwijs ik naar de architectural concerns. Simpel voorbeeld: er is een non-functional requirement en die is het systeem moet 24/7 beschikbaar/online zijn. Vervolgens ga je naar je deployment diagram, en dan ga je bepalen van hoe ziet mijn infrastructuur eruit. En dan heb je te maken met high availability, en dan ga je een active-active setup neerzetten bijvoorbeeld: en dat doe je vanwege die architectural concern.

> Doe je dat voor jezelf of communiceer je dat ook naar developers?

Ze weten dat, maar informeel. Ze weten het omdat je er continu mee bezig bent. Wat je vaak ziet, is je maakt op voorhand een AD, om die blueprint te hebben. Ik ben niet van de lijvige documenten en alles in den treure uitschrijven, maar je moet een blueprint hebben. Je fase 0 maak je die blueprint, en besrpeekt hem dan met de ontwikkelaars. Maar dat is vaak een een eenmalige actie. Sommige ontwikkelaars lezen dat AD, maar dat gaat vaak het ene oor in het ene oor uit. Na een tijdje komt er een nieuwe ontwikkelaar bij het project, en dan vertel je hem van daar vind je de documentatie. Maar je gaat niet verifieren of hij dat ook echt gelezen heeft. Die ontwikkelaar gaat vragen stellen op een gegeven moment van joh ik kom dit tegen waarom is dit zo? Van we hebben ooit die beslissing genomen, waarom is dit zo?

> Is het in de praktijk fijn als alle concerns en hun ontwerpbeslissingen worden vastgelegd in een rationale?
>
> 36:00 min

Nee, in tegenstelling, die is niet fijn. Ik heb op een project gezeten waar alles tot in den treure werd uitgeschreven, en toen dacht ik echt als ontwikkelaar, ik kan hier niks mee, dit is zoveel detail. Dan lees je eerder helemaal niks. Laat me gewoon mn ding doen.

> Wat we willen doen is die belangen laten terugkomen in het hele proces van ontdekken van inconsistenties. Heel vaak worden belangen goed geidentificeerd. Daarna worden die belangen

> in modellen/diagrammen vastgelegd. Er is dan een soort van drift, van belangen naar die modellen, als de belangen veranderen. <<gelul over methode>> . Welke vragen zou je stellen per cel in de matrix?

De eerste vraag die ik zou stellen is: wat gebeurt er als we er niks mee doen? Ik zie het, hoor het en lees het, dit is schijnbaar belangrijk. Wat gebeurt er nou als we er niets mee doen? Als het een echt conflict is, dan moeten we het blijkbaar wel doen, dus dan is het antwoord dat er iets gebeurt als we het niet doen. Factoren zoals kosten en tijd zijn erg belangrijk. Feitelijk, elk project heeft een doelstelling. En op het moment dat het dus ter sprake komt dat die doelstelling in gevaar komt, dan hebben we een issue te pakken. De doelstelling is het belangrijkst. Op het moment dat we dan echt een conflict hebben, dan is namelijk de conclusie 'dat kan niet': ik moet of deze laten varen of deze laten varen. Dan moet je op zoek naar alternatieven. De volgende vraag is, welke alternatieven heb ik? Om nog even terug te komen op zonet; als we het drag and drop voorbeeld hebben, wat is dan het meest gebruiksvriendelijke alternatief om deze functionaliteit te maken [met deze technologie]. En je gaat dus op zoek naar alternatieven in beide richtingen [in die matrix] die niet meer met elkaar in conflict zijn.

> Als kosten, tijd, en alternatieven er niet zijn, dan is het de vraag welk belang gaat dan van tafel?

Op het moment dat er een belang van tafel wordt geveegd, haal ik dan mijn projectdoelstelling nog? Als ik mijn projectdoelstelling dan niet kan halen; dan hebben we een stopper te pakken. Je moet ook realistisch zijn; en dan moet je gewoon zeggen jongens; het kan niet.

> Dus zou zo'n matrix dan ook een goede communicatietool zijn? Om aan stakeholders te laten zien. Ik heb hier een framework van acht bij acht cellen, deze cel is gewoon rood. Dus we kunnen niet door.

> We willen misschien een tool bouwen, online, die geldt als een soort instrument waarmee de architect inconsistenties in zijn architectuur kan ontdekken, maar die ook gebruikt kan worden als een soort wiki. Het zou dus mooi zijn als verschillende partijen kunnen inloggen.

Waar ik altijd heel sceptisch in ben, als het gaat om dit soort web tooltjes. 9/10 keer zeg ik wat is de meerwaarde bovenop een documentje en een spreadsheetje. Ik heb altijd evernote bij me. Als ik iets zie wat belangrijk is schrijf ik dat op, en dat werk ik dan later wel uit. Vervolgens ga ik kijken wat staat er allemaal in mijn evernote, en dat ga ik doorvoeren in mijn SAD. Dat is 1, ik snap ook wel, die web interface is lekker centraal en je kan mensen er naartoe linken. Alleen wat je vaak ziet is dat je hier een tooltje heb en daar een tooltje, mensen komen om in de tooltjes. Het wordt interessant op het moment dat het een toevoeging is op een bepaalde toolset. Pak JIRA voor ticketing, pak Confluence als wiki, en pak Bitbucket voor je sourcecode. **Als dit nou een plugin is, die in deze suite werkt, zeg ik helemaal goed!**

> Als we belangen willen prioriteren, hoe zou je dat dan doen?

Altijd terug naar doelstellingen. Ik hou niet van MoSCoW, want alles is Musthave. Wat je moet doen, is gewoon een waardering aankunnen. Je hebt een projectdoelstelling: en het gaat om de bijdrage aan die projectdoelstelling. Als iets bijdraagt aan die doelstelling. Ik zou het gewoon met kaarten doen, de pokermanier. Er wordt een referentie gemaakt van een bepaalde use case, die dan bijvoorbeeld "3" kost. Relatief aan die use case worden andere use cases in geschat, en wordt er gezegd, deze kost "1". Als iedereen het eens is, dan is er niks aan de hand. Als er verschillende kaarten worden getrokken (we gebruiken de Fibonachi reeks voor de cijfers) dan ontstaat er discussie en moeten de developers gaan praten. Dit zou je ook voor de prioritisering van de belangen kunnen doen.

Wat ik nog interessant zou vinden: kijken of er een bepaald pratoon is te ontdekken in belangen die niet met elkaar stroken en dus inconsistent zijn. Is het dan mogelijk om te kijken of je ook een bepaald soort patroon kan ontdekken in acties die er mogelijk zijn om die inconsistenties te omzeilen? Dus als je als het ware een architecture inconsistency pattern krijgt waarmee je inconsistenties tussen bepaalde concerns kan verhelpen die vaak voorkomen.

Expert 2, EXPERT INTERVIEW, 19-1-2016, 17:30

Over inconsistenties: dit soort inconsistenties wil je al zo vroeg mogelijk kwijt, die wil je gewoon niet hebben. Het ene zal veel makkelijker op te lossen zijn dan het ander. In een zo vroeg mogelijk stadium in je traject moet je al zorgen dat je zoveel mogelijk fouten eruit haalt.

Kromme van Bohm: naarmate je verder in je ontwikkelingen komt wordt het kostbaarder om fouten te herstellen. Die kosten gaan exponentieel omhoog.

1. Wat is je rol als SA?

Een team dat onderhoud doet van legacy applicaties van bedrijf X. Architectuur is al gegeven. Moet je voor zorgen dat alles er zo goed mogelijk inpast. Liefst mogen er geen nieuwe technologieen inzitten, want het is een eindig traject. Het mag niet teveel geld kosten, dat is de belangrijkste voorwaarde. Binnen dat speelveld moet ik mijn werk doen. Daarnaast doe ik met name veel ontwikkelwerk. Dat is voor mij een voordeel omdat ik op die manier contact blijf houden met alles wat er speel. Ook vind ik het erg leuk om met mijn 'knieen in de modder te staan'.
Als architect ben ik dus betrokken bij het in stand houden van de omgeving en de voorwaarden die er zijn om het systeem in de lucht te houden, zijn bedrijfskritische systemen. Het is voornamelijk COBOL: weinig nieuwe instroom, animo is niet zo groot, veel oude mannetjes.

2. Kom je wel eens inconsistenties tegen in de praktijk en hoe ga je ermee om?

Je komt ze altijd tegen in de praktijk, en je komt ze vooral tegen in de praktijk omdat je te maken hebt met de requirements van de gebruikers. Die requirements zijn niet altijd samenhangend. Soms wil een gebruiker iets dat in het systeem niet kan, of dat gebruiker A het zo wil maar gebruiker B het zus wil. Als je daarmee te maken krijgt, moet je dat gewoon uitpraten. Als requirements in tegenspraak zijn met elkaar dan moet je zorgen dat je er een grootste gemene deler uit krijgt, en voor die tijd kan je niet eens aan de slag. Je kan niet 2 kanten op ontwikkelen, en dan halverwege stranden.

V: maar bij grote projecten kan ik me voorstellen dat je niet altijd doorhebt dat die belangen tegenstrijdig zijn, of de modellen tegenstrijdig zijn? Wordt dat gemonitord of gecheckt?

We hebben normaal gesproken 2 informatieanalisten en die filteren heel veel dingen er bij de analyses al uit. Die praten veel met gebruiken ook, en die hebben veel kennis, waardoor ze veel inconsistenties glad kunnen strijken in een vroeg stadium. **Dan heb je nog inconsitenties tussen wat men wil en wat er met het systeem kan, en daar loop je dan tijdens je ontwikkeltraject tegenaan.** In de regel los je dat zelf op als ontwikkelaar. Je doet het niet op eigen initiatief, maar je gaat wel overleggen: gebruiker, analist. Lastige is wel dat als je je gebruikers/kenners/analisten kwijtraakt, dat je lastiger problemen kunt verhelpen in zo'n situatie.

Bepaalde inconsistenties worden er dus uit gehaald door informatieanalisten, degenen die ze missen kom je tegen in je ontwikkelwerk, maar die probeer je zo goed mogelijk op te lossen door te overleggen met je team en gebruikers die kennis hebben van zaken?

Correct. Bij ons is het team niet zo heel erg groot. De informatieanalisten leveren hun analyse op, daarna wordt er een schatting gemaakt en dan gaat er een ervaren ontwikkelaar aan de slag met het functionele ontwerp van dat probleem. In de meeste situaties komen dat soort zaken bij het functioneel ontwerp wel boven tafel. En dan wordt het gesignaleerd en dan gaat het terug naar de informatieanalist of terug naar de gebruiker. De functionele ontwerpen worden over t algemeen wel op papier gemaakt. Vaak worden deze als basis gebruikt, voor nieuwe dingen. Als dat niet goed is gedaan, kan je op basis

van wat je nodig hebt wel een architectuur maken ad hoc. In de praktijk ga je goed nadenken over hoe het er in gaat passen, dus dan kom je al wel veel problemen tegen.

> Waar komen nieuwe concerns/functionaliteit vandaan bij jullie? Ik heb het idee dat er nogal vaak nieuwe dingen worden toegevoegd.

Dit zijn vooral dingen die moeten gebeuren. Schadevrije jaren bij autoverzekeraars. Daar heeft men gepoogd om er eenduidigheid in te brengen, dit moesten wij aanpakken. De belastingdienst heeft ook soms nieuwe voorwaarden gesteld waaraan voldaan moet worden, dan moeten we die systemen aanpassen/onderhouden: nieuwbouwen of verbouwen.

> Als een ontwikkelaar een inconsistentie tegenkomt, dan gaat hij een overleg plannen zeker als het belangrijk is. Hoe wordt het besproken?

Wij hebben met ons team van ongeveer 10-12 man hebben wij geregeld teamoverleg en daar kunnen dit soort dingen aan de orde komen, ook sochtends hebben wij een korte standup sessie. Daar kan je problemen aankaarten. Het idee is dan dat je na dat overleg even met zn tween gaat zitten. Maar heel vaak gebeurt dat natuurlijk ook wel ad hoc dan wil ik gewoon direct even overleggen.

> Heb je daar een voorbeeld van? 20:00 min

We hebben nu een nieuw project, waarbij privegegevens van 7 jaar en ouder verwijderd moeten worden uit de systemen. Aangezien die systemen vaak data hebben die langer dan 7 jaar is opgeslagen, en ook privacy gevoelige gegevens, daar kom je op een gegeven moment uit -
Je moet een plan opstellen, dit hebben we nu in deelplannen gedaan, omdat het te groot is om in 1 x te doen, we zijn nu begonnen met een onderdeel van het grote plan: het gaat om de relatieadministratie. Binnen die relatieadministratie heb je allerlei tabellen die opgeschoond moeten worden, maar die mogen alleen onder bepaalde voorwaarden opgeschoond worden. Je kan je voorstellen dat als persoon A een verzekering heeft gehad in het verleden, maar die heeft hij nu niet meer, dan kan je je voorstellen dat die gegevens best opgeruimd mogen worden. Maar als persoon A een verzekering heeft gehad, maar hij heeft daar nog een uitkering op (door AOW), dan wordt het al lastig en wil je die gegevens behouden.

Je kunt in grote lijnen bedenken, ‘we kunnen het zo oplossen’, maar daarna ga je het in detail bekijken en dan pak je de databaseontwerp bij, want daarin staat beschreven hoe de tabellen aan elkaar hangen, maar je moet ook over het randje heenkijken, maar je moet ook weer andere systemen en relaties bekijken, en zo kom je uiteindelijk toch aan een hele reeks van dingen aan die je gewoon moet weten voordat je aan de slag gaat. Dan maak je een functioneel ontwerp waarvan je denkt, daar zit het meeste wel in. Dat verspreid ik dan naar mensen waarvan ik denk die weten er wel van. Dan krijg je commentaar en dat verwerk je, daarna ga je het zelf nog eens herzien. Ik ga dan kijken, hoe kan ik zo goed mogelijk tot een oplossing komen - die ook werkbaar is - want het moet ook nog geregeld draaien dus het mag niet te lang duren, je moet zeker weten dat je niet teveel weggooit. Dat probeer ik allemaal op een rijtje te zetten. Op elk punt waar ik vastloop probeer ik iemand anders in te schakelen.

> Worden inconsistenties geclassificeerd?

Wij hebben een service delivery, die is verantwoordelijk voor het inde lucht houden van de systemen, als er problemen zijn dan krijgen zij het als eerst voor hun kiezen. Dat kunnen bestanden zijn die er niet zijn, of problemen in de onlineomgeving waardoor de gebruiker niet kan werken. Zij gaan dan uitzoeken wat er aan de hand is, maar daar kunnen ook problemen uit voortvloeien, die komen dan bij ons terecht. Die komen dan bij ons op het lijstje te staan. Ook krijgen we onze opdrachten eigenlijk binnen

via de gebruikersorganisatie, die krijgen een request for change binnen, die worden dan ingeschaald op belangrijkheid door de gebruikersorganisatie, en op een bepaald moment wordt er dan een release planning opgesteld. In die release gaan we die, die en die problemen oplossen. Er wordt ingeschat hoeveel tijd dat kost: raakt het andere vlakken, wat moet er voor gebeuren? Dan komen wij weer in beweging.

> Ben je tevreden met de manier waarop het detecteren van tegenstrijdigheden en het afhandelen daarvan gaat?

Ik probeer te zoeken naar manieren waarop dat efficienter kan, want je raakt soms de grip erop een beetje kwijt. Er is volgens mij bij ons niet goed op belegd: er zijn teveel mensen die zich er mee bezig houden, met het classificeren - eigenlijk het **administreren** van die problemen. Dat is normaal gesproken de taak van de SA, maar in ons geval is dat een vreemde eend in de bijt. Want dat is bij ons niet gewoon dat jij dat zou doen, meeste registratie vindt plaats bij de SD club, en de andere problemen dat ligt bij de gebruikersorganisatie. Die gaan in overleg met de projectleider, uiteraard in overleg met ons. Eigenlijk is hij de gene die moet zorgen dat die planning passend is.

> Het zou dus efficienter zijn als 1 of 2 mensen (1 rol) die problemen verzamelen, identificeren, classificeren, administreren?

Ja, dat is zeker waar. Maar wees bewust van het feit dat je zoekt naar een rol en niet naar een persoon, dat is altijd lastig als een persoon wegvalt. [Het gaat om de taken die je doet.]

> Worden bij jullie de ontwerpbeslissingen gekoppeld aan de inkomende concerns/belangen/RFCs?

Je hebt natuurlijk standaarden en richtlijnen, constraints, die zijn vrij hard, die zie je altijd terug in de architectuur. De architectuur is al heel hard gegeven, en je verandert er niet zo heel veel aan, tenzij je echt moet. Als voorbeeld:

In een vorig bedrijf moest een mainframe gekoppeld worden aan een vergunningssysteem, dat systeem dat gaf vergunningen uit aan financieel adviseurs. Ik moest het mainframe koppelen aan dat vergunningssysteem. Ik heb daar toen een Java applicatie voor gemaakt, die de URL in de gaten hield, en bij een wijziging downloadde hij een bestand, en gooide dat in een database, die database werd gekoppeld aan het mainframe. Ik kreeg dat niet voor elkaar om het vanaf het mainframe te doen, en dan word je gedwongen om een Java applicatie te maken. Dan doe je het in dit geval om het geheel gewoon werkend te krijgen, en dan kijk je niet meer zo van ‘past dat nu wel in het landschap’. Het is wel gevaarlijk, ondanks alle goede documentatie ging dat programmaatje down vlak voordat ik weggging omdat ze de URL hadden gewijzigd. Daar was het wel op voorbereid, want er was een configuratiefile (invoer, wat moet je doen) bij, maar de ondanks erg goede documentatie (al zeg ik het zelf) las niemand dat. Hoe goed je iets ook documenteert, toch doet niemand daar dan wat mee. Dat komt omdat het een vreemde eend in de bijt is (past niet in de architectuur), dus niemand keek er naar. Dus 2 dingen zijn belangrijk: hoe goed je documentatie ook is, als 1) niemand het leest, en 2) niemand het bijhoudt is het zonde.

Maar in dit geval was de beslissing om die Java applicatie te maken ondanks de architectuur die het mainframe is in feite, omdat het gewoon niet anders kon - ik had er geen andere mogelijkheid voor.

> Hoe kunnen we concerns prioriteren?

Dat is in feite het wezen van het hele architect zijn. Je hebt te maken met verschillende stakeholders en hun concerns, en als een stakeholder zegt van ik wil dat je die concern addresseert want dat is mijn allerbelangrijkste concern. Dan houdt het op, dan kun je niet anders. Hoe los je dat nou op? Je kan wel tegen een stakeholder zeggen: 'je kan wel zeggen dat deze belangrijk is, maar als ik deze niet kan doen dan kan jouw concern ook niet geimplementeerd worden'. Hangt heel erg af van de concersn waarmee je te maken krijgt op dat moment. **Ervaring is heel belangrijk, en ook herkomst van de concerns**. Degene die betaalt bepaalt zeggen wij altijd. Uiteindelijk zal een klant het laatste woord willen hebben.

Wij gebruiken ook RCDA. Je probeert de risicos en de kosten van je concerns zo goed mogelijk vast te stellen, maar dat blijkt lastig. Je hebt natuurlijk veel verschilende partijen, en zoveel verschillende belangen. Tijd is indirect kosten. Tijd kan meer zijn dan alleen kosten. Soms refereert het naar deadlines die echt gehaald moeten worden. Voorbeeld: ziektekostenverzekeringen: die moesten op 1 januari gewijzigd zijn, maar op december kreeg je die wijzigingen binnen. Dan is tijd een erg belangrijke factor, en niet per se 'tijd is geld'. Volgende factor: inpasbaarheid binnen het systeem, inpasbaarheid binnen het bedrijf.

> Stel je voor dat we 2 belangen hebben die elkaar overlappen. Hoe zou jij kunnen detecteren of daar een probleem kan ontstaan in de software.

Ervaring is weer erg belangrijk: als je ervaren bent, ken je de systemen van binnen en van buiten. Dat geldt ook voor de informatieanalisten die wij hebben. Ervaring is belangrijk in dit verhaal. Door die ervaring weet jij als er wensen of wijzigingen komen van buitenaf, waar die dan gaan clashen. Hoe je dat dan weet, dat is lastig om te zeggen. Ik kan niet tegen jou zeggen hier heb jij een heel systeem, en jij krijgt allerlei wijzigingen, ik zou niet zomaar een kruismethode bedenken waarbij je met 2 rijtjes je dat kan aangeven, als je daar een methode voor kan vinden hou ik me warm aanbevolen.

> <uitleg methode> @ 45:00 min

Eigenlijk ga je dan een beetje naar een manier toe van Agile werken, daarin ga je uit van veel kortere cycli. Wij zijn waterfall gewend, maar in Agile ga je user stories in veel kortere iteraties werkend proberen te krijgen. Ook vindt hier veel prioritisering plaats. Dat werkt verlichtend in de zin van dat je kunt beginnen met de requirements die eenduidig zijn, en dat wat nog niet helemaal duidelijk is dat wordt in de loop van je project door voortschrijdend onderzoek duidelijk. In de loop van het project wordt het steeds duidelijker waar het nu gaat knijpen. Daar gebruik je ook dat pokeren: gebruikers, testers, informatieanalisten, business owners. Ze weten allemaal iets van het systeem, en de scrum master is eigenlijk een soort van architectenrol op dat moment, hij is verantwoordelijk voor het beheren van het tempo. Bij een dergelijke werkwijze loop je nooit zomaar tegen grote problemen aan, omdat die al door het team opgelost kunnen worden.

> <Uitleg invoegbaarheid van methode>

Scrum is gewoon een invulling van Agile. En wat je zegt doe je impliciet al in Scrum, stel je eigenlijk in het begin van elke sprint vast van wat gaan we deze sprint opleveren, want je moet altijd een eindresultaat opleveren in een sprint, en het moet iets werkends zijn. En je hebt te maken met gebruikers van kennis van het systeem hebben. Uiteindelijk ben je met het hele team verantwoordelijk voor een eindresultaat wat gewoon moet werken, en als je als team daar niet uitkomt. Dan is het dus belangrijk om onderling veel te praten, en problemen te bepraten. Dat kun je op verschillende manieren oplossen. Uitstellen, compenseren, enzovoorts. Architectonische conflicten heb je veel minder last van. Ook omdat je al prioriteert, en omdat je afspreekt hoeveel je gaat doen.

> <Uitleg middenstuk methode>

Daar zie ik wel waarde in. Agile is natuurlijk ook geen tovermethode. Je geeft goed aan, in het begin van je traject stel je andere concerns vast dan gedurende het traject. In het begin van het traject kun je bijvoorbeeld hebben in welke omgeving ga ik draaien, en hoe ga ik dat oplossen. Dan is het misschien wel goed om in het begin te constateren van let op, als we hier tegenaan gaan lopen hebben we een probleem. Dan moet je dat vroegtijdig op gaan lossen. Eigenlijk moet je je methode invoegen in de zero-iteratie van je agile traject.

> Zou je de matrix ook bij kunnen werken tijdens volgende iteraties op het moment dat er nieuwe concerns bijkomen?

Ja dat kan zeker. Want du moment dat er een nieuwe user story bij komt, kan dat mogelijk leiden tot problemen. Als je dan een overzicht hebt van de user stories en de problemen die dat op kan gaan leveren, want eigenlijk doe je dat een beetje, dan kun je heel snel zien van 'o, let op, ze hebben hier lopen klooien'.

> Zou je er toegevoegde waarde inzitten?

Zeker. Waar ik in geinteresseerd ben is dat ik aan het einde van de rit een systeem op kan leveren waar de gebruiker tevreden mee is. Als er in dat systeem nog allerlei dingen niet goed zijn dan krijg ik dat op mijn brood. En dat wil ik niet.

> Hoe zou ik deze methode in je dagelijkse manier van werken kunnen inpassen zonder al te veel weerstand - zodat het je niet veel overhead oplevert?

Uiteindelijk moet dit je gaan helpen om tijd te besparen, dus als je [als SA] nuttig gebruik kan maken van de methode dan levert het je tijd op. Ik denk een spreadsheet een prima techniek om dingen in kaar te brengen, en dan is het van belang wat je in de x-as zet en wat je in de y-as zet, en waar je de sluitingen maakt. Ik zou me kunnen voorstellen dat je bepaalde punten uit user stories haalt, en daarvan zegt dat je er bepaalde problemen in voorziet: je kan je voorstellen dat als je in de interface van 1 systeem gaat zitten rommelen, dat een ander systeem daar last van gaat krijgen. Dat is een heel duidelijk voor de hand liggend probleem. Maar zo zijn er misschien ook wel andere problemen, die veel moeilijker te detecteren zijn. En daar ben ik nou juist naar op zoek. Ik wil graag zien: "jij gaat nu dat doen, maar dat dreigt helemaal verkeerd te gaan, en waarom dat nou verkeerd gaat, dat zien we later wel". Ik wil graag dat oranje of dat rode lichtje krijgen van 'opletten'. Uiteindelijk moeten we het wel zelf oplossen, wat voor methode je ook gebruikt, hij lost niks op.

> Die methode moet zo goed mogelijk een architect helpen om er op een gestructureerde manier met zijn team over na te denken dus?

Moet licht en gemakkelijk in het gebruik zijn, spreadsheetje kent iedereen, daar kan iedereen mee omgaan. Een webtool is op zich nog geen slechte oplossing, daar kan je heel veel kanten mee op. Een probleem is bijvoorbeeld dat mensen thuis gaan werken en niet bij de netwerkschijven kunnen. Dit heeft als gevolg dat we dan zo'n spreadsheet moeten doormailen etc etc. Dan is een centraal online tooltje wel handig; elke laptop heeft een browser.

> Als we concerns centraal stellen in de methode, hoe kunnen we deze dan goed verpakken?

Ik wil in 1 oogopslag zien wat er aan de hand is. Dus de concerns nummeren en dan en die nummers in een opzoek-index zetten is niet handig bijvoorbeeld, omdat je dan veel opzoekwerk moet doen. Je hebt natuurlijk altijd van allerlei toetertjes en belletjes waarmee je van die hover veldjes kan maken enzo. Op

het moment dat je zoiets gaat maken, moet je mensen erbij roepen en zeggen wat vind je ervan. Prototyping ben ik een groot voorstander van omdat je dan preceis kan laten zien wat je doet en wat het wordt. En dan kan een gebruiker gewoon zeggen van ‘goh ik vind het prachtig, maar ik zou er hier of daar wat anders aan doen’. Prototypen is het meest waardevolle instrument wat ik heb.

Wat je kunt doen voor de evaluatie is alle mensen die je gaat interviewen bij elkaar in 1 lokaal te doen, een soort brainstormsessie, en dan geef je een sessie op een elektronisch bord. Waarin je de methode uitlegt en vertelt hoe het werkt, en dan kunnen mensen er rustig op los branden. Je kan gewoon een business case opstellen, waarbij je een simpele webapplicatie pakt en waarvan iedereen wel een beetje weet hoe het werkt, en je zodoende de methode kunt uitleggen aan de hand van een voorbeeld.

Denk eraan: uiteindelijk zal je niet iets heel statisch moeten hebben, want architectuur is heel dynamisch. Je moet willen blijven leren.

Expert 3, EXPERT INTERVIEW, 21-1-2016, 10:00

Uit die modellen wordt code gegenereerd, die per definitie consistent is, want die komt uit dezelfde generator. De consistentie tussen code en model, die is er altijd.

1. Je rol als SA, weet ik al een beetje. Hoe worden inconsistenties gemonitord/gecheckt?

Het team dat ik aanstuur is 6 man groot, en onderling moet je natuurlijk dingen afspreken, en dat zetten we op een wiki, dat zijn de werkafspraken. Ook bijvoorbeeld hoe je je code schrijft. Waar dat mogelijk is, proberen we die regels ook te borgen in checks die we automatisch kunnen doen. Dus elke regel source code die je maakt moet dus aan die checks voldoen. En anders kan een computer dus al afkeuren, van deze regel is niet consistent met de rest. Maar dan praat je wel alleen over de taal waarin je programmeert. Op dat niveau proberen we de code consistent te houden en dus ook hopelijk dat iedereen elkaars code kan lezen. Je inentingen kunnen zo en zo verlopen, dat kunnen we gewoon checken, maar of je nou logische namen gebruik voor variabelen, dat kan een computer niet checken. Daar hebben we ook een soort codereview voor, dat iemand anders je code nog een keer bekijkt voordat het echt af is, dat is meer de code kwaliteit waarborgen.

We hebben ook een boyscout rule van vind je iets dat niet helemaal klopt zet het dan even recht als je het vindt. Dat niet de volgende persoon zich daar weer aan ergert en dat het op een gegeven moment er een cultuur heerst waarbij het er niet uit maakt hoe het er uit ziet, dat is ook een stukje cultuur: als je een stukje tegenkomt wat niet goed is en het kost weinig moeite om het recht te zetten, zet het dan ook even recht. Een soort Agile ontwikkelings regel. En je zou denken is het niet een risico als je continu wat kleine dingetjes loopt aan te passen voor het zelfde geld maak je iets kapot, maar we zorgen ook dat we genoeg tests hebben op onze code zodat je niet zomaar iets kapot kan maken, en dat in de geautomatiseerde tests allemaal naar voren zou moeten komen. Dus dat is moeilijk om dat helemaal waterdicht te maken. Maar we hebben wel zoveel tests, dat we dat in vertrouwen kunnen doen. Testen zorgt ervoor dat je met meer vertrouwen de code durft aan te passen. Als ik iets doe wat niet kan zorgt die test ervoor dat je het zelf weet, voordat de klant - of veel andere mensen erachter komen.

Het niveau waarop wij programmeren is niet dat we allemaal schermen programmeren, nee we programmeren een knop, en dit soort logica, en al die knoppen moeten natuurlijk wel op een soort consistente manier werken. Dat is toch met de hand geschreven code, en we gaan naturulijk geen code kopiëren en plakken dus we maken gelijk wel utility's. Wat we nog wel willen doen is een copy-paste detector inbouwen zodat je ook ziet deze twee stukken lijken zoveel op elkaar, die zijn waarschijnlijk gekopieerd van elkaar. Dat zijn kwaliteitsmetrics die je aan kan zetten, die hebben we voorlopig nog niet aanstaan - wil ik wel graag. Dat die het al ziet, van deze twee stukken lijken zo op elkaar, hier had waarschijnlijk een stuk weggehaald moeten worden. Wat dus ook onderhoudbaarder wordt. Je krijgt dan dus juist niet dat je code consistent moet zijn, maar dat je code juist uniek moet zijn.

> Je gaat dus ook niet continu monitoren voor consistentie, maar probeert de inconsistenties er snel uit te halen?

Op het moment dat twee stukken source code consistent zijn, lijken ze blijkbaar veel op elkaar dat het misschien juist een helperfunctie moet zijn die je dan op 2 plekken aanroept en die juist specifieke dingen daar moet doen. Consistentie is natuurlijk een middel, en een doel is natuurlijk onderhoudbare code. Het belangrijkste doel is onderhoudbaarheid. Daar maak je ook werkafspraken voor. Buiten de source code om hebben we heel weinig documentatie. Dat vinden we niet nodig of proberen we tot een minimum te beperken, de ervaring leert ons dat dat soort dingen altijd achterlopen. En dus off to date raken.

> Dus door de comments in de source code wordt het e.e.a. begrijpelijk gehouden, en doordat jullie een team zijn van 6 man?

We zijn gegroeid van 2 man naar 6 man, dus ja we zijn ook veranderd in de zin van dat onze ontwikkelaaanpak: met 2 man had ieder zn eigen taken en wist je waar je mee bezig was maar nu hebben we er voor gezorgd dat we meer een scrumteam zijn, met standup meetings enzo.

> Dus als ik het even kort samenvat proberen jullie die inconsistenties in de architectuur grotendeels te checken door checks, tests, en goede werkafspraken onder de de teamleden.

Als je vaak met elkaar werkt, dan kijk je ook bij elkaar en leer je ook van elkaar en dan doe je dingen ook op dezelfde manier.

> Vorig jaar vertelde je dat jullie niet helemaal verantwoordelijk zijn voor de veranderingen of binnenkomende requirements, het contact met de gebruikers.

Dat zijn de ontwerpers, die zitten hier tegenover, die bedenken wat wij moeten bouwen.

> Die requirements komen dus binnen bij jullie, nou kan ik me voorstellen dat die requirements wel eens tegenstrijdig kunnen zijn. En ook daar wil ik zo veel mogelijk over leren. Hoe monitor je dat die niet tegenstrijdig zijn?

Dat is moeilijk want je moet zoveel mogelijk in je hoofd hebben want je moet zoveel mogelijk in je hoofd hebben want dit stukje informatie conflicteert met dit en dit verhaal. En inderdaad, er was een zoekveld en een tabel eronder, en als je dan vanuit het zoekveld pijltje naar beneden deed, dan wil je dat de focus naar de eerste regel gaat. Stel dat er een nieuwe requirement komt van stel dat je het zoekveld opent, dan moet er een typehead komen met de laatste drie dingen waar je op gezocht hebt. Wat er dan gebeurt is dat de cursor naar de type ahead gaat en niet meer naar die eerste regel, en wat is nou belangrijker?

Soms lees je het ontwerp tegen en dan bedenk je het al, soms ben je aan het implementeren en dan bedenk je het, en andere moment heb je het geïmplementeerd en dan doe je het, en dan denk je van he wacht: dit betekent dus ook dit. Of je ziet beide dingen tegelijk verspringen: je hebt dan snel door dat het een bug is.

> 14:30 Op kleine schaal lukt dit misschien wel, maar hoe doe je het op grote schaal? En wat zei je: kan je tijdens het ontwerp erachter komen, tijdens het implementeren, en nadat je geïmplementeerd hebt?

Liefst wil je er zo vroeg mogelijk uit komen, zodat het zo weinig mogelijk impact heeft, en dan hebben de minste mensen er last van, het liefst zou de ontwerper er al achter willen komen.

> Vind je dat het optimaal gaat?

Ontwerpen zijn altijd wel of niet, beter uitgewerkt of juist niet, moesten we soms dingen verzinnen, en in de regel wordt hier aardig formeel gewerkt. In het grote geheel is soms lastig, je hebt gewoon een grote bak met requirements, terwijl de structuur ertussen moet uiteindelijk ook weer helder zijn, dat als er iemand nieuw in het team is dat hij dan van requirements naar gerelateerde requirements kan springen. Om het hele plaatje helder te krijgen. Maar goed bij elke organisatie is dat lastig, om goed te borgen wat waren nou de requirements allemaal. Ook als er dingen achterhaald zijn of er zijn dingen weg. Daar

gebruiken we een soort wiki voor, maar nog niet in de vorm van een wiki ofzo. Dat moet eigenlijk wel gestructureerd worden. Als een soort kennisbank. In die fase zitten we nog niet echt. Uiteindelijk ligt daar ook wel verantwoordelijk bij de ontwerpers.

> De structuur in die requirementsdatabank is nog niet optimaal?

Die is nu op een ontwerp gericht van implementeer dit maar, doe nu dit stapje, voeg nu dit eraan toe, terwijl als iemand het over een jaar terugleest dan wil hij niet zien 'doe nu dit, doe nu dit, doe nu dit', nee hij wil zien hoe het nu is. Of hoe het nu had moeten zijn.

Ik denk dat het verstandig is om niet allemaal upfront design te doen, maar je wil juist dat het incrementeel gaat. Juist waar mogelijk zoveel mogelijk ontdekken.

> Dus continu blijven verbeteren is beter dan in 1x opleveren? Zijn dan de testen ook makkelijker te schrijven?

Bij scrum wil je het liefst stories hebben; dit soort functionaliteit gebruikt eerst dit en dan dit, dan gebeurt er dit: dan is het iets wat je kunt bouwen op deze manier, je kunt het borgen op die manier en je kunt het testen op die manier. Zolang die story nog valide is moet die test ook nog staan.

> Kan je wat vertellen over hoe jullie nu testen?

De code die we nu schrijven testen we automatisch. In de zin van unit testen, waarbij we elke regel code unit testen, op basis van deze feature had er moeten zijn. Daarnaast hebben we nog wat integratietesten, als de applicatie zo is opgezet, dan moet de gebruiker dit en dit kunnen doen door de applicatie heen. Als we het dan opleveren, is er nog een testteam. Die op hun manier gaan testen, met de TMAP methode.

> Hoe worden prioriteiten aan requirements gehangen?

Dat zouden we niet zelf beslissen, maar dat zouden meer de managers en de projectleiders moeten zijn. Wij kunnen wel dingen signaleren en daarover beslissingen afdwingen. Doel je nu op Functionals of op NFRS? *Eigenlijk naar beiden op zoek.*

We willen gewoon een goed werkend systeem hebben richting 2019 en de kwaliteitseisen die dan van toepassing zijn willen we helemaal halen, qua performance en onderhoudbaarheid. Maar goed, we willen nu ook heel graag snel nieuwe functionaliteit ondervinden, werkt het goed werkt het lekker. Dus ook functioneel proberen we wat meters te maken. Wat we wanneer oppakken.. Allebei tegelijk dus eigenlijk.

> Op welke manier prioriteren jullie die elementen? Intuitie? Bepaald systeem?

In principe is het dus intuïtie, en ik denk dat het doel is risico's of bepaalde dingen waardoor we geen product hebben op de gewenste datum, dat we die naar voren trekken. Tenminste dat is neem ik aan waarmee de managers de beslissingen nemen. Je wilt weten dat het performt, en je wil weten dat het schaalt, en je wil weten dat het gebruiksvriendelijk is, en daar wil je het liefst nu al ervaring mee opdoen. En niet als je pas alles gerealiseerd hebt, en denkt: dit had nooit gekund. Maar er zit niet een methodologie achter, iets wat ontwikkeld is.

> Ik zie risico, schaalbaarheid en performance vaak langskomen, en dat op basis daarvan ook vaak functional requirements worden ingeschat op -

Kan dit nog wel, ja precies. Dat is natuurlijk de taak van een architect, of de architectuur aanpassen op basis van de gewenste functionaliteit, of de functionaliteit aanpassen aan de architectuur, of wat is het hoogst haalbare. Wij moeten dus eigenlijk zeggen dat als we in 2019 live willen gaan, en onze architectuur kan dit niet aan, maar die van onze concurrent kan dat wel aan - dan hebben we de verkeerde architectuur gekozen, maar goed we proberen de architectuur zo goed mogelijk in ons inziens zo goed mogelijk te krijgen.

> Hoe kan je jullie model het best omschrijven?

We gebruiken Powerpoint om data en dingen over te dragen, maar uiteindelijk komt het in onze database terecht. Deze componenten bevatten deze eigenschappen of attributen, personen, prijs enzo. Die dingen leggen we allemaal vast. Op basis daarvan maken we ook flowcharts van dit zijn processen in ons systeem VB: als je dus een artikel verkoopt, dan moet het bij die persoon in zijn bakje komen. Dat zijn wat meer visuele diagrammen. Daaruit wordt werkende software gegenereerd.

> Ik kan me voorstellen dat er ook wel eens flowcharts gemaakt worden die inconsistent zijn, komen jullie dat ook wel eens tegen? Als de flowcharts tegenstrijdige informatie bevatten..

Het model kan best inconsistent zijn; dat er iets ontstaat wat niet gegenereerd kan worden. Stel je hebt een voornaam en een achternaam en de volledige naam is de voornaam plus de achternaam, stel dat de achternaam afhankelijk is van de volledige naam dat je een soort cirkel krijgt, dat kan natuurlijk niet. Dat kan niet voorkomen. Op dat moment is het model dus invalide en de generator zou dat moeten idealiter al moeten zien. Of de modelleeromgeving. Op die manier wordt dat gemonitord. Het kan op hele complexe manieren verkeerd gaan, dat als je een constructie gebruikt hier, waarvan de ene constructie hier gebruikt wordt en de andere constructie valt in die andere omgeving dan kan het op een gegeven moment zijn dat je dat nooit hebt gegenereerd dus dan hebben we niet ingebouwd hoe dat zou moeten. Of we kunnen technisch of wiskundig bewijzen dat het niet kan en dat het dus nooit goed gaat. We willen wel beperken wat er mogelijk is: als iets een proces is, dan kan het geen locatie zijn. dat kan je mooi inbouwen in het model.

> Een groot voordeel is dat jullie dat model driven engineering gebruiken. Maar op het gebied van inconsistenties is het dus wel een groot voordeel. Kan ik me voorstellen? 26:30

Ja dat is zeker waar. De generator speelt daar dus ook een grote rol als controlerende factor, monitor.

> Interessant wel. Jullie zijn het eerste bedrijf waar ze MDE gebruiken.

Juist leuk toch. Heb je meerdere dimensies in je onderzoek!

> Ik ben ook bij bedrijven geweest waar ze werken met diagrammen, modellen en views, op basis waarvan dan de de ontwikkelaars aan het werk worden gezet.

Als die diagrammen niet kloppen, of nieuwe inzichten bij de ontwikkelaar, die moeten weer terug inde diagrammen terugkomen, of dat doen ze niet. Of dan komt de tester, en die test volgens de oude specs, en dan zijn er weer allemaal dingen afgesproken.

Bij ons speelt het natuurlijk ook wel dat als de ontwerpers iets bedenken, en wij implementeren het dan, dan kom je erachter dat het niet helemaal klopt, of dat het niet helemaal kan. Dan komt er een requirement van doe het dan maar zo en zo, en als er dan een tester nog volgens de oude specs gaat testen dan.. dat moet wel kloppen zeg maar, dus het ontwerp moet wel. Een tester mag dus niet testen

op de oude requirements terwijl er al nieuwe requirements zijn doorgevoerd. Of als er onderling al even afgesproken is van dan moeten we dit even zus en zo doen, en dat is niet doorgevoerd. Dat soort dingen kom je nog wel eens tegen, maar niet heel veel gelukkig. Het is ook wel handig dat het een wikipagina is, dus die hoeft alleen geüpdatet te worden het is niet zo dat er een nieuwe versie van het ontwerp gemaakt moet worden, moet geaccordeerd. Het is gewoon platte tekst en die pas je aan. De tester weet dus ook dat hij in de wiki terecht kan. Hij zal vaak eerst naar de ontwerper gaan en zeggen van; klopt dit wel?

> Dus ook inderdaad een manier van samenwerken, cultuur, wat je zei.

<uitleg methode, werking, doel>

Voor die matrix, zet je dan dezelfde concerns in de horizontale en de verticale as neer, dus dat je kijkt per twee dingen? En hoe die bij elkaar komen. Je kunt denken aan functional dingen de beveiliging en performance. En als je die ook aan de andere zet, dan raken ze elkaar natuurlijk. VB: extra beveiligingsmaatregelen inbouwen, gaat ten koste van de performance dus als ik continu wil checken of de gebruiker die nu iets opstuurt of die nog wel ingelogd is, maar qua beveiliging wordt het er wel weer extra veilig van en dan moet je een afspraak maken van doen we dat wel of doen we dat niet. Dat is dan zo'n puntje, je kunt volgens mij dezelfde dingen op dezelfde as neerzetten, en als die dingen bij elkaar komen, dan zouden het aandachtspuntjes worden.

> De vraag is dan natuurlijk van hoe, en wanneer neem je zoiets door? Wat je niet wil is dat een team een enorme extra berg overhead krijgt.

We proberen natuurlijk in sprints incrementeel op te leveren.

> Op basis van eerder onderzoek kom ik tegen dat die evaluatie vaak plaatsvindt aan het einde van een cyclus, op een latere fase.

En dan is er geen tijd of geld meer om terug te zetten of terug te gaan.

> Precies, dan is het heel kostbaar. Want dat is mijn volgende vraag, hoe zou je die concerns verpakken?

Het zijn eigenlijk lastige elementen om weer te geven. Wij hebben wel gezegd qua codekwaliteit het belangrijkste is beveiliging, en dan performance, en dan onderhoudbaarheid, en noem maar op. Ja goed, de functionaliteit moet te realiseren zijn.

> Maar als je zo'n stukje als performance pakt, hoe praten jullie daarover dan? Want het heeft natuurlijk een soort van heel veel verschillende uitwerkingen en aspecten. Als ik het over performance in die matrix heb, dan is dat natuurlijk een lastige term om te beschrijven.

Performance geldt natuurlijk voor de databasetechnologie, de laag er tussen, het netwerk, het dataverkeer, de browser, dus op al die punten moet de schaalbaarheid en de beveiliging en de schaalbaarheid weer goed zijn. En op al die vlakken zijn weer andere concerns te bedenken, waar je dan weer aan gedacht moet hebben, en waar je afspraken over moet hebben gemaakt.

> En denk je dat het waarde toevoegt als je dat gestructureerd weergeeft?

Op die manier haal je wel pijnpunten naar boven, van deze twee concerns conflicteren met elkaar, en nou goed, als je dan daar op dat moment volgens de huidige inzichten afspraken over maakt hoe dat

nou zou moeten en dat toetst in de praktijk en dan kun je nieuwe afspraken maken op dat vlak, voegt wel meer toe dan als je die lijst gewoon plat houdt met, dat je de lijst niet met elkaar afstemt, dan wordt het meer een checklist. Het wordt wel een klein vakje en dan zou je er misschien meer een getalletje in moeten zetten en dan wordt het meteen weer een soort checklist…

Het moet ook in de hoofden van de mensen zitten, als die elkaars code bekijken. Maar goed, dat is het nadeel van de architectuur,

> Ik probeer zoveel mogelijk van de praktijk meenemen. In wezen is het ook een beetje een doel van mij om de methode een soort dan manier van communiceren te maken, misschien wel als onderdeel of toevoeging aan de wiki. waar zowel testers als ontwerpers maar ook architecten en ontwikkelaars op kunnen teruggrijpen. Waar wringt het ? Wat voor afspraken hebben we daarover gemaakt? De gemaakte afspraak inzichtelijk te maken? De uitkomst van elk probleem, elke overlap, die moet inzichtelijk gemaakt worden ook.

Die moet je tussen de oren krijgen van de mensen, maar je kan het wel op de wiki zetten, maar als de wiki een lange lijst van honderd punten bevat dan leest niemand het ook meer. Dus het moet wel hiërarchisch gestructureerd zijn, en als je hier mee bezig bent dan moeten de voor jou relevante dingen naar voren komen. Met tools kan je sommige dingen checken, maar andere dingen niet. Wat is te checken met een tool, en wat niet? De automatisering is ook heel belangrijk. Als je dus nieuwe afspraken hebt door gevoerd, stapje voor stapje. dan moet je ook meten zijn we daar nu klaar mee, hoeveel werk is dat nog om dat door te voeren. dat is ook moeilijk inzicht. een deel maken we taken van en die zetten we op de backlog om te doen. en ander stukje moeten we functioneel wat aan veranderen dus moeten we een stukje code herschrijven dan nemen we dat gelijk mee. Als je allemaal verbeteringen aan je architectuur aan het inbouwen bent dat is moeilijk aan de buiten wereld te verkopen dat zelf even stil ligt. daar hebben ze hier in ieder geval begrip voor want ze willen vooral een goed product hebben. zo zijn we hier overgestapt van javascript naar typescript omdat dat makkelijker is: bij ons leveren we dan een tijdje even wat minder functionaliteit op.

En dat proberen we dan weer zo te doen dat we in kleine stapjes zichtbaar kunnen maken wat wat we aan het doen zijn.

> Jullie zijn van javascript naar typescript gegaan?

Er waren genoeg technische redenen omdat te doen.

> Ik kan me voorstellen dat een nieuwe ontwikkelaar wil weten waarom die beslissing is gemaakt. wordt dat ergens vastgelegd of niet?

Nee dat wordt niet ergens vastgelegd, we zouden hem in geuren en kleuren kunnen vertellen wat we aan het doen zijn. Tegenwoordig is het zo doe scrum doe agile en probeer het team zo hecht mogelijk te krijgen, hoe het niet te formeel want ja formeel is ook lastig. maar goed.

Stel dat deze methode uitgewerkt wordt, wat zou er voor nodig zijn om deze methode goed in de dagelijkse gang van zaken te laten passen.

Er moet een goede afweging zijn van wat het kost om de tool bij te houden en wat levert de tool ons op. Die waarde moet groot genoeg zijn. We pakken niet zomaar - tijd is een belangrijke factor. Weinig tijd kosten, veel inzicht verschaffen. Geen grote hoeveelheden administratief werk, of een tool die zelf achterloopt qua data. Probeer ze min mogelijk tools te hebben dus een integratie in onze bestaande tools. Dat zijn dingen die gewoon blijven werken.

Tijdens zon standup meeting?

Wat doe je tijdens een standup meeting. Wat je doet, wat je gaat doen, Wat voor problemen je tegenkomt. Ik kan iets gesignaleerd hebben de vorige dag wat ik de volgende dat bij de standup wil bespreken die afspraak zie dat die niet nageleefd wordt, en dan zou ik achter ene pc zelf recht gaan zetten.

een methode wordt altijd weer veel werk. binnen een scrum team moet je denken aan het feit dat iedereen een beetje een gelijkwaardige rol heeft. Iedereen moet die tool gaan bijhouden, iedereen moet hem kunne gebruiken. het moet ook in de werkwijze van iedereen passen.
als iemand een code review doet, dan zijn er dingen van ja naar welke punten kijk je nou naar, dat zijn misschien wel dingen die uit deze tool naar voren kunnen komen. Bij het opstellen daarvan. Kan best zijn dat we tijdens het toevoegen van een nieuwe requirement kijken met die matrix van op welke punten speelt het allemaal.

maar goed je moet niet duizend punten hoeven checken, je moet wel de relevante punten checken. daarom is die prioriteit ook wel belangrijk die je eraan hangt.

Expert 4, EXPERT INTERVIEW, 21-1-2016, 15:00

1. Kun je kort je rol als architect beschrijven?

2. Op welke manier worden inconsistenties in de architectuur gemonitord en gecheckt? 7:30

Ik zit nu in een project over levertijden. We willen de klant beter informeren over levertijden. Dat wil je eigenlijk overal doen. Dus op meerdere plekken in de webshop. En dus ook op meerdere plekken in de architectuur. Daar zijn eigenlijk 4 architecten bij betrokken: we hebben ruwweg gezegd dit zijn de interfaces tussen onze systemen. Ieder gaat dan gewoon een systeem uitwerken. Dan ga je die uitwerken en dan binnen je eigen domein ga je nooit zo diep in de architectuur dat je inconsistenties tegenkomt. Wat je wel ziet is dat als je een afspraak had over een interface, en je gaat even een week uit elkaar, en je gaat die uitwerken, dan heb je gewoon interpretaties die zijn echt anders. Dan heb je gewoon inconsistentie over het geheel genomen, van die 4 architecten waar we dan een discussie mee hebben.

We proberen heel erg te focussen op het benoemen van deelgebieden die ieder van ons dan uitwerkt, om te voorkomen dat we allemaal het zelfde doen [of op elkaar moeten wachten]. En waar we dan telkens achter komen en waar de discussies vaak over gaan is juist over die inconsistenties. We hebben enerzijds de gaten, dan kun je gerust zeggen 'het niet inconsistent, het is gewoon nog niet klaar', daar heb je veel discussie over, van hoe gaan we dat samen oplossen.
Anderzijds, hadden we toevallig het vorige uur een discussie over, een specifieke interface, waar mijn aanname was dat veld x erin zou zitten, terwijl hun aanname was dat veld y erin zou zitten. Met allebei ook echt compleet goede redenen. Waar we echt een lange discussie over hadden. Dat ontstaat gewoon natuurlijk tijdens het proces, en dan zien we het uiteindelijk wel weer terugkomen, juist omdat je weer bij elkaar komt. Bewust om door te spreken wat heb jij uitgewerkt, wat heb jij uitgewerkt, etc. Op dat moment focussen we ons erop. Daarmee sluit ik niet uit dat er ergens anders niet meer zijn, maar op dat moment heeft dat de focus.

Hoe wordt dat dan gecheckt? Op basis van..? Dat wordt ad hoc gedaan, toch wel op basis van intuitie en ervaring?

We hadden allemaal wel de meeting goed voorbereid, allemaal een plaat voorbereid. Dan zie je gewoon, het past niet meer. Je ziet gewoon, het klopt ff niet meer. Daar hebben we dan een best wel forse discussie over gehad, en daar wordt het ook beter van. Waarom kies je nou hiervoor? Ik heb eigenlijk zo gekozen. Dan krijg je een gesprek.

Een mooi voorbeeld was ook vorig jaar, een ander doet ook functioneel en non-functioneel, en dan oke we hebben afspraken gemaakt. In de checkout (afrekenen) vragen we hoeveel voorraad is er nog, want dan kan je beter voorspellen of de klant het vandaag of morgen nog krijgt. Toen hebben we dat gebouwd, bleek dat toen het live ging die oproep veel te vaak werd gedaan waardoor het systeem omviel. Functioneel klopte het dus gewoon, maar nonfunctional waren ze eigenlijk nog gewoon vergeten dat de load zoveel hoger zou worden. In dat geval heb je ook gewoon een inconsistentie te pakken, tussen je functionele eisen en de je non-functional [specs].

Hoe worden inconsistenties bij jullie geclassificeerd? Hoe bepalen jullie welke vervolgstappen je met een inconsistentie doet? Kun je dat aangeven?

Ik denk impliciet. Kijk we hadden deze sessie [vergadering voor dit gesprek] omdat we vonden dat er iets aan moest gebeuren. Als niemand het belangrijk had gevonden, hadden we het niet gedaan. Het is niet dat we documenteren we hebben deze inconsistentie gevonden hier moet nog iets gebeuren, dat niet. Wat we wel eens doen in de voorbereiding van een project. Vanuit de architectuur begin ik altijd met een soort ideaalplaat als we dit oplossen gaan we dat hier doen en dat hier doen enzovoorts, en dan denken we ja, maar dat is leuk, maar ik wil het volgende week hebben. Dan denken we kunnen ook zo en zo en zo doen, en dan hebben we het over twee weken en dan werkt het ook prima. Maar dan spreken we wel af dat we het ook over een half jaar netjes gaan maken. De architectuur schrijft voor, dit moet zo, in de realiteit is het gebouwd op een andere manier, dus inconsistent - aan de architectuur. Maar met een hele concrete afspraak met we gaan het wel oplossen. Dat is vooral een lijst van wat we wél moeten doen. Niet echt een classificiatie nee.

> Ik kan me ook voorstellen dat developers als ze een inconsistentie tegenkomen ze die vaak zelf fixen, ofwel als ze een inconsistentie tegenkomen die niet belangrijk is ze die gewoon laten zitten.

Als je het zoekt in de sfeer van bugs - want in mijn optiek is een bug ook een soort van inconsistentie - die classificeren we wel. Daar hebben we echt low-medium-high-en critical bugs, daar wordt ook verwacht dat critical meteen opgelost moet worden en high mag max twee weken duren, dus daar hebben we een soort van classificatie. Dat is wel redelijk low-level, m.a.w. dat is wel erg in de code, en dus ook niet helemaal architectuur-niveau. Een bug is een vorm van inconsistentie waarbij je opdracht niet consistent is met de voorgeschreven syntax van je taal.

> Op welke manier handel je zo'n inconsistentie af? Ben op zoek naar de factoren die doorslaggevend zijn in zo'n gesprek.

Uiteindelijk is het vooral een inhoudelijke discussie, maar waar we in deze setting vooral aan elkaar gewaagd zijn. Wel luisteren we heel goed naar elkaar. "Waarom dacht jij dit?" Openstellen en discussie voeren. Dat helpt vooral omdat je in een setting van architecten zit, dan heb je toch een beetje hetzelfde belang. Andersom zit je met bv business stakeholders, en dat is eigenlijk toch wel hetzelfde, gewoon open het gesprek in gaan. En argumenten op tafel leggen, wil het iets complexer of groter worden is het toch wel van maak nou ff inzichtelijk met een plaatje of met een schets hoe dan ook wat nou het probleem is, wat is nou die inconsistentie, zoals jij dat noemt. Wat klopt er nou niet, en belangrijker nog, wat gaat er mis doordat het niet klopt. Want als er niks misgaat dan hoeven we niets op te lossen. Als we iets zien dan moet het ook wel echt iets betekenen. [Het gevolg dus echt]. Maar inderdaad ook op langere termijn, wat zijn de schades van dat dit niet klopt.

> pakken jullie ook alternatieven aan: het is nu zo, dat is problematisch. Hoe kunnen we dat oplossen, en van elke oplossing, wat zijn daarvan de gevolgen?

We hebben dat probleem van die performance, eigenlijk kun je dat maar op 1 manier oplossen, dat is door een extra database layer te introduceren, dat is best wel prijzig dus dat kunnen we nu niet doen, dan lossen we het nu niet op.. [dan heb je weer een afweging tussen prijs en …] dan is de keuze ook geweest, tegen de business, van we kunnen garanderen dat 90% van alle calls die krijgen een antwoord, wat in ons geval wel heel laag is, want dat zijn gigantisch veel klantbezoeken. De business accepteert dat dan. Maar als we over een tijd meer gebruik willen gaan maken van die feature moeten we wel die nieuwe databaselayer introduceren, dat hebben we dan ook afgesproken voor over een paar maanden. In de tussentijd is het gewoon de afweging, het is nu te duur, dus we accepteren het even.

> Dus wel een goed voorbeeld van een inconsistentie die wordt getolereerd inderdaad.

Precies. Uiteindelijk is het wel de business die dat te duur vindt, maar dan wel accepteert. Beslissing om zoiets te tolereren is ook wel afhankelijk van de betrokkken stakeholders en wat voor economische waarde daaraan hangt. Er zijn gewoon zaken die van een business case afhingen inderdaad. Dat kost ons dan zoveel conversie, omzet of geld. Zij moeten die afweging echt maken en dan uiteindelijk zeggen van ik accepteer die 10 procent. Of, nog anders het kan ook zijn dat ze zeggen: “het kost ons heel veel geld, maar als we dit nu niet oppakken, dat gaat meer geld opleveren dan dat dit kost”, dus als we dit uit gaan stellen, dan.. - het kan ook een relatie met een totaal ander onderwerp zijn, als we dit nu niet doen dan gaat het langer duren voordat we hierop verdienen.

> Zijn op dit moment dingen, op het gebied van het oplossen van dit soort inconsistenties, waar je niet tevreden over bent? Denk je dat het optimaal gaat?

Zeker niet altijd, soms duurt het gewoon erg lang. Soms duurt het ook lang om tot de kern te komen, dan zie je wel dat er iets mis zit, maar wat zit er dan eigenlijk mis. Dan kost het gewoon tijd om erachter te komen wat er mis is. Dat is vaak als er al gebouwd is, anders hadden we het van te voren wel gezien. We hadden het zo bedacht maar het werkt toch niet helemaal. Hoe komt dat nou? Het is ook nog wel eens moelijk om een besluit te vinden, wie kan er nou over beslissen? Heti s niet altijd heel direct aan een specifieke business of project toe te wijzen, tja er gebeurt iets met performance, dan weet je niet wie er precies verantwoordelijk voor is. Waar het nu best lastig is, is: 1) hoe gaan we het besluit nemen, en 2) vinden waar het nou precies inconsistent is.

> Precies, waar overlappen nou twee belangen.. -

Het zijn ook niet altijd twee dingen he, het kan ook een ophoping van tien dingen zijn, waar het hier ergens begonnen is en waar het zich hier pas uit, dus het is ook niet altijd daar waar je het resultaat ziet dat daar de oorzaak is. Het is ook die root cause analyse proberen terug te toveren, en daarmee proberen op te lossen.

> Houden jullie ergens bij hoe je ontwerpbeslissingen zich verhouden tot de input, tot de requirements?

Ik denk niet dat we dat hebben. Wat er hier vaak gebeurt is dat de business een wens heeft, dan luister je die aan, dan ga je hem interpreteren en dan kom je met een ontwerp, en dan kom je samen en dan ga je het er over hebben, maar het is niet zo dat we een hele lijst van requirements proberen bij te houden. Maar dat zou in die gevallen wel helpen, ben ik met je eens. Op het moment dát het misgaat denk ik ook wel we hadden misschien toch wel die tijd moeten nemen om het iets gestructureerder aan te pakken.

> En dan zie je vooral waar je beslissingen vandaan komen -

Beslissingen zijn bij ons vaak, dat je na de meeting even samen komt en zegt, hier waren we het over eens, als je het er niet mee eens bent dan laat je het even weten. Het is niet gestructureerd vastgelegd. Waar we het wel gestructureerd vastleggen is in de NFRs. Voor ieder systeem hebben we wel echt een document waar vaststaat ‘zoveel responsetijd’, ‘zoveel berichten’, ‘zoveel storage’.

> Ook met redenen erbij?

Ja, zeker bij security en performance hebben we redenen erbij. We moeten zoveel responsetijd omdat we zoveel calls per uur krijgen. Dus niet per se de echte business reden, maar meer we weten dat er zoveel load aankomt, dus daarom nemen we een beslissing.

Waar wel heel vaak dingen streng worden gedocumenteerd, is bij embedded software, bv printerbouw. Of medische apparaten. Daar heb je gewoon een hele andere werkwijze. Wij werken aan een webshop, is er eeen keer iets kapot, dan is het vervelend, maar gaat er niemand dood. Als we down zijn is het ook wel echt heel ruk, en dat zijn we gelukkig ook nooit, maar dan valt er niet direct iets om. Bij ons is het wel zo, hoe verder je in de keten gaat hoe erger het wordt, kijk kom je echt in het warehouse terecht, zon systeem moet echt niet een uurtje stilstaan. Situatieafhankelijk is een belangrijke factor.

> Als we concerns als first class objects gebruiken om een framework te creeeren waarin de architect en zijn team kunnen kijken om mogelijk inconsistenties te ontdekken, hoe zou je dit dan doen?

Waar voor ons altijd een hele belangrijke afweging zit die vrij makkelijk te maken is, is responsetijd. Iedere miliseconde die je sneller bent levert gewoon meer verkoop op, omdat mensen neit hoeven te wachten. De afweging kan je ook makkelijk doen, wensen zijn vaak uitbreiding van de functionaliteit, dus ik heb een paar en ik voeg er iets aan toe, en dat heeft effect op response tijd, dus daar kun je gewoon heel makkelijk afwegen. Ik voeg een feature toe, dat levert wat op, maar mn responsetijd gaat erop achteruit. Maar ik lever dus ook in; die dingen staan vrijwel altijd haaks op elkaar. De lastigere is daarbij 'het gemak'. Ik voeg een nieuwe feature toe aan een gebruiker, vindt hij de weg dan nog wel. Dat is moeilijk meetbaar te maken, wat is daar het effect nou van. Ik denk dat het dus heel vaak op die assen zit: "dit is wat ik functioneel zou willen, en dit is wat het technisch voor mij betekent". Technisch onderling heb je ook weer tegenstrijdigheden. VB: We zouden graag minder storagegroei willen, [want dat kost geld], maar tegelijkertijd willen we alle data die over jou als klant te vinden is, want dan kunnen we beter personaliseren. **28:00 min**

Wat heel erg zou helpen is een soort. De 20 meest voorkomende conflicten, en dan het antwoord op response tijd en zulk soort features. Of neem hardware/virtualisatie en schaalbaarheid. Dus meer die perspectieven tegen elkaar uitzet.

> Het zou natuurlijk mooi zijn als je dingen tegen elkaar uit kon zetten, maar ook kon inzoomen. Dus dat je op schaalbaarheid kan klikken en dan inzoomt om de details te zien als je dat nodig acht. Hoe zou je die categorieen op de assen prioriteren?

Ik zou het heel lastig vinden. Het is wederom situationeel, het is echt welke view je pakt en op welke manier je kijkt, technische of functioneel. Ik denk wel dat je aan alles een kostenplaatje kunt hangen, dus kostentechnisch kun je alles wel sorteren, al is dat in een matrix nog wel moeilijk. En de belangrijkste is natuurlijk altijd risico. Neem responsetijd weer wat voor ons een hele belangrijke is, degene waar daar op de snijvlakken waar het daar rood is, daar loop je een groot risico op de responsetijdterugloop, die zou ik dan graag wel bovenaan zien. Voor storagegebruik gelden misschien wel weer totaal andere regels.

> Dus eerst je perspectieven kiezen, en dan ga je dus op basis van context prioriteren? Correct. Een idee van mij was om eerst die concerns te verzamelen, om daarna de prioriteit eraan te hangen, en dan de belangrijkste meenemen in de matrix. Precies.

Voor deze specifieke feature is dit gewoon relevant, je hoeft niet altijd naar die matrix van honderd bij 100 te kijken. Je pakt eruit wat je nodig hebt. Dat zou helpen ja, als het licht was, dat je niet alles hoeft te checken. Dat dwingt je ook niet om alles al helemaal in te vullen,

> Maar het dwingt je wel op een bepaalde manier de tradeoffs te moeten bekijken. Waar ik onzeker over ben, is of die hypothese "waar pijnpunten ontstaan, daar kunnen inconsistenties optreden" juist is.

Ik denk dat dat zeker juist is, ik denk dat ze meestal rond die gebieden ontstaan, vaak waar je een afweging maakt kies je iets, en iemand anders zal de afweging anders maken.

> Hoe zou je die concerns kunnen verpakken?

Ik zou denk ik gevolgen proberen te identificeren. Als je de as pakt en je zou een soort typering van features hebben [dus wat voor categorieen features/acties zijn er] en je zet dat af tegen een lijst van NFRs. Stel je pakt een type UI aanpassing, die zal op het snijvlak met storage weinig snijvlak hebben, maar hij zou op het snijvlak met responsetijd behoorlijke gevolgschade kunnen hebben. Het scherm wordt complexer dus de responsetijd loopt terug. Dat je echt een beetje tastbaar maakt, wat betekent dat snijvlak nou, dat zou ik proberen zo concreet mogelijk te maken. Die termen op de assen zijn al zo abstract.

> In hoeverre, als we die methode bouwen, in hoeverre moet de methode dat aanleveren, of in hoeverre moet de architect dat doen?

Wat ik zou proberen is wel om een standaard te hebben, maar hem wel zo veel mogelijk levend te houden. Oke als je nu een specifieke use case pakt, en je gaat nu deze feature bespreken, vul die matrix voor dat geval, waardoor het geheel weer uitgebreider wordt. Ik kan deze er ook nog bij verzinnen, dat er een levendig iets wordt. Ik zou proberen om hem niet in 1x helemaal vast te zetten. Ik heb nu weer een nieuwe use case, dan veranderen er ook weer een boel belangen. Of je zegt ik werk aan iets standaards: voordat je met iets nieuws begint, pak je die matrix er even bij. Gebruik wat er staat, maar vul het ook vooral aan. Dat die continu wordt aangepast en verbeterd. Je wilt een soort initial fill hebben. Abstract kun je zelf [jasper] wel een heel aantal voorbeelden bedenken die mensen aan het denken kunnen zetten. Maar het is juist belangrijk dat zo'n methode specifiek voor een bedrijf gemaakt kan worden. Dat je zegt van nu ga ik helpen met jullie om hier iets van te maken wat helemaal Company X eigen is. Onze snijvlakken zullen hele andere voorbeelden opleveren dan bij bijvoorbeeld company Z, of company Y.

> Wat is er voor nodig om de methode goed in je dagelijkse praktijk te laten passen?

Tijd is een lastige factor. Alles wat een beetje naar bureaucratie neigt, formulieren dat soort dinge, daar wordt een beetje afstand van genomen, toch zie je wel dat een heel aantal dingen, zoals security en NFRs die moet je gewoon invullen voordat je een nieuwe feature begint. En dan hebben we het niet over de kleine verbeterinen maar over grote dingen. Als we hier nu zien dat we een heel werkbaar model krijgen waarbij we met hele simpele korte checks even een aantal aandachtspunten boven tafel kunnen halen om het gesprek te starten met stakeholders of andere architecten van is hier aan gedacht is hier aan gedacht etc. dan is het ook niet meer belemmerend. Niemand is er op tegen om een framework in te vullen, als je er ook daadwerkelijk iets uithaalt waar je iets mee kan. En dan moet je zien te bereiken. Dus als je echt helpt die dialoog te starten, dan denk ik zeker dat er iets in zit.

Wat moeilijk is is wel functionals tegen de functionals afzetten. Je hebt natuurlijk een hele lange lijst van functional concerns in je bedrijf, misschien wel miljoenen. Die kun je ook tegen elkaar afzetten. VB: mijn doelstelling is ik wil meer marge maken, en aan de andere kant ga ik dus een feature bouwen die een betere verkoopprijs bepaalt. En aan de andere kant zegt iemand, we gaan iets soepeler met die leverancier om. We gaan hem iets meer ruimte geven om zn inkoopprijs te verhogen, maar dan gaat ie ook beter leveren. Maar dan doe je dus precies het tegenovergestelde, dat is ook iets dat bijt, en dat is natuurlijk heel moeilijk. Om dan te vinden - NFRS zijn eigenlijk een soort van limiterende lijst, daar kun je d'r best wel 30-40 van verzinnen maar of deze nieuwe feature deze nieuwe feature bijt, dat is natuurlijk heel moeilijk te bepalen.

> Daar kan die architecture rationale dus bij helpen, dat je dus vastlegt waarom je bepaalde beslissingen maakt. Dus als je dan ziet we hebben 2 nieuwe initatieven, en dan kan je een soort van backtrace doen.
>
> Functionals kun je vaak vanuit een perspectief zien, dus data, informatie, deployment, etc. Deze kan je natuurlijk ook tegenover elkaar uitzetten, en met de methode mee leveren, dus dat je zegt: jongens heel simpel, als je functionals wil gaan checken, zorg dan dat je een perspectief in acht neemt.

Oh dat is wel een goeie, dat je gewoon inzoomt op een bepaald deelgebied.

> Op welke andere manier kun je de methode gberuikeN?

Online soort wiki, dat doen we met de NFRS ook, en dan is het nog wel de vraag of het gebruikt wordt, dat is vaak het lastigst, dat niemand er dan op kijkt.

H.2 Evaluation: Case Studies

Expert 3, CASE STUDY, 03-03-16, 10:00 uur

Eerste vraag is eigenlijk, in wat voor fase zitten: hebben we al een bestaand product, of gaan we een nieuw product maken. Als je nog moet gaan beginnen, dan is het meer op conceptueel vlak, heb je al een product dan ga je zoeken in je systeem en documentatie. *Je hebt natuurlijk ook nog een middenweg he? Dat je een bestaand product hebt, en je moet bijbouwen maar daar wel rekening mee houden dat je een bestaande architectuur hebt.* De methode doelt eigenlijk op de architect steun en support geven tijdens zijn architectuurproces/ontwikkelproces, met focus op inconsistentie.
Dan ga je naar de situationele factoren. Dat is eigenlijk een manier om te bepalen hoe je als bedrijf deze methode kunt doorlopen. En dat hangt dus af van een 14-tal factoren die ik heb geidenticieerd. De architect wordt wel aangeraden om zelf te kijken wat vind jij belangrijk. Voorbeelden zijn tijdsdruk, stabiliteit van de omgeving. Ander voorbeeld is teamgrootte. Hoe groter je teams, hoe systematischer je moet werken tijdens de methode.

Zo ontstaan er drie profielen, en de architect beslist dus op basis van die profielen hoe die de rest van de stappen eruit komt te zien. Daarna is het de vraag hoe, waar en hoeveel stakeholders je wil gaan betrekken. Als je met vragen komt te zitten over inconsistenties, kun je stakeholders betrekken om conflicts te solven etcetera. Als je de methode wat systematischer uitvoert, adviseert de methode om grotere groepen stakeholders te betrekken. Laat situationele factoren zien: zoveel mogelijk situaties toepasbaar is. Dan scope: functionele scopeafbakening. De methode is eigenlijk een template, de invulling geschiedt door de methode uit te voeren en door de architect zelf. Volgende fase, je kiest de perspectieven die belangrijk zijn om te evalueren naar aan leiding van je project goal. Van die perspectieven kies je dan concerns in de stap Gather Concerns, de concerns die op dat moment handig zijn. Ik moet er inderdaad bij zeggen, de methode neemt aan dat de concerns al geidentificeerd zijn. Deze methode probeert niet over te nemen wat er in de praktijk al gedaan wordt. Een vereiste is dat de architect al de concerns al heeft.
Daar geef ik een uitleg voor, maw je kan elke techniek kiezen die je wil, als je zelf maar een prioriteit hanteert. Als er maar rangorde ontstaat in de longlist. Concerns worden eerst verzameld, geprioriteerd en daarna gespecificeerd. Daarna is het belangrijk, dat je een design rationale ontwikkelt. Als er al een design rationale is, is het belangrijk dat ie als input voor het proces geldt, of als hij er niet is, dat de architect en zijn team dat aanmaken. Voor de belangrijkste concerns, identificeer hoe die zijn verwerkt in de literatuur. Waar creëer je de link tussen je belangen en hoe ze zijn gemaakt in de architectuur. Een hele pragmatische architect doet dat misschien wel ad-hoc, of uit zijn hoofd, als je wat gestructureerder werkt ga je dat misschien wel documenteren.

Daarna ga je de matrix bouwen: het idee is dat je de matrix cyclisch doet en dat je nieuwe concerns pakt wanneer nodig. Idee is heel simpel, er worden dus perspecieven gekozen, er worden daarna concerns ingeladen, en nu heb ik ervoor gekozen om mijn dataperspectief tegenover mijn dataperspectief te zetten. Concern tegenover concerns is een grijs vakje bij wijze van spreken. Maar je data ownership tegenover je rollback mechanisms, daar kan het bijvoorbeeld gaan wringen. En dan zie je dus dat het team op deze manier inzicht kan krijgen in waar die concerns overlappen. Het vullen van de matrix doe je in deze stap. De stap erna zegt eigenlijk van, is er een overlap, dus raken ze elkaar. De preconditie voor inconsistentie is dat er overlap is. De eerste stap is: kijk, en ga na of er overlap is tussen twee concerns. Want als er overlap is tussen twee concerns, dan is er ook overlap tussen twee diagrammen, (die die concerns addresseren). Dan zeg je, het is eigenlijk niet erg dat er overlap is, als er dan maar geen conflict optreed

- Stakeholders kunnen mogelijk meehelpen met het prioriteren van die concerns.
- Bepalen of er een ripple effect is, (of er negatieve consequenties zijn als je het oplost), hangt zelf weer van de oplossing af.
- Je kan een inconisstentie oplossen, negeren

- Je kan een inconisstentie uitstellen _ wel markeren dan
- Je kan hem omzeilen, een workaround verzinnen, maar zonder hem op te lossen
- Je kan hem ook nog verbeteren, verzien van commentaar.
- Het zou een handvat moeten zijn voor een architect om te helpen met oplossen.

Bespreken keuzeregels

- Participant heeft moeite met het begrijpen van ignore en improve ten opzichte van resolve. *Postpone begrijp ik. Uitstellen, maar we moeten het wel doen. Bypass begrijp ik ook, resolve begrijp ik. Ignore is gewoon: we weten het maar we negeren het. Maar waarom kies je dan voor improve dan? In welke stiuatie zou je voor improve kiezen?*
- Je kan stappen terug doen, na de methode, om nieuwe concerns te kiezen, nieuwe perspectieven, of helemaal naar het begin waarin je meteen een nieuwe scope kiest.

Uitleg tool

Wat is de exacte invulling van een concern? Dat is contextspecifiek. Op vat voor hoogte bekijk je een concern? Naam, requirements, stakeholders, perspectief, add custom field.

“Volgens mij ben ik wel heel enthousiast eigenlijk, ik zie hem in ieder geval wel, laat ik dat zo zeggen, in ieder geval toegevoegde waarde. Met name die prioritering van concerns, en dan volgens die perspectieven, tegen elkaar afzetten, dat eh, cool”. Ben alleen wel benieuwd hoe je dit werkend gaat krijgen.

Ik heb de methode eigenlijk gedesigned met als uitgangspunt dat ik niemand een strobreed in de weg wil liggen, maar dat ik wel voldoende houvast en guidance wil geven. Vandaar ook de modulaire stappen. Sommige bedrijven zullen niet graag actief op zoek gaan naar inconsistenties, en dat deel is dan ook minder bruikbaar dan inconsistency management. Daar ligt dan de focus op.

33:00 min Uitwerken voorbeeld

Er moet een systeem gebouwd worden, voor de beveiligersvereniging Nederland. Alle beveiligers moeten een speciaal pasje krijgen. Een zogenaamd beveiligerspasje (BeveiligersID (BID)), is een fysiek pasje dat elke beveiliger krijgt. De reden dat ze een pasje krijgen heeft te maken met de wet nachtbeveiligingsaansprakelijkheid, een discotheek is verantwoordelijk voor alle beveiligers aan het werk op die locatie. Zo’n eigenaar moet dus weten voor wie hij aansprakelijk is. Ook: vaak geschiedt beveiligen op basis van zwart werk, mannen die dubbele diensten draaien etc. Wat moet er komen: een systeem wat enerzijds de uitgifte van de BIDs sfaciliteert. En anderzijds moet er een systeem komen wat registreert welke beveiliger op welke locatie/festival is op welk moment.

Er is gekozen om een portaal te maken, voor registratie van BID’s, en daar hangt een database aan met ID’s. Ik ben een beveiliger, en via dit portaal kan ik een BID aanvragen. Uiteindelijk gaat er via een uitgiftesysteem een fysiek pasje naar de beveiliger. En daar komt dan een fysiek pasje uit. Dus elke beveiliger in Nederland moet naar dat portaal.

Dan hebben we aan de andere kant nog de locaties/discotheken/festivals: daar komen de beveiligers aan, en die moeten geregistreerd worden. Die moeten in- en uitklokken. Registratie van aanmelden, en registratie van afmelden. Dit is een beetje in de basis het systeem: er zijn een aantal concerns die je meteen kan noemen. Het eerste concern dat je kunt noemen: in NL zijn 180.000 beveiligers snachts actief. Daarvan begint 95% tussen 22 en 23 uur. Dat betekent dat ik 95% van 180.000 binnen een uur moet verwerken. Dat is 1 concern. Andere concern, een keiharde eis, dat ik hier 5 jaar data moet bewaren: een constraint. Een concern is dan dat die database heel groot gaat worden: aantal aanmeldingen per dag, x2, x220 dagen in het jaar, x5 jaar. Levert belachelijk veel entries op in de database. Dan heb ik nog een concern: het moet wel gebruiksvriendelijk zijn: beter gezegd: de

beveiliger werkt voor SECUNITE, en de SECUNITE heeft bij elk fesitval een eigen toegangscontrole systeem. Die beveiliger checkt helemaal niet in in ons systeem, dat doet hij bij SECUNITE, en niet met zijn Beveiligerspasje (BID) maar met zijn SECUNITE gegevens. Ik krijg hier, mijn SECUNITE checkin binnen. De stroom wordt gesplitst: beveiligers die zich er wel aan houden, checken in met hun ID, en de gebruikers die zich er niet aan houden checken in in hun eigen systeem. Nou is het belangrijk dat het zo makkelijk mogelijk is voor SECUNITE om zich aan te sluiten op ons systeem. Dit moet dus een interface zijn die ze makkelijk in hun systeem kunnen implementeren. Dat ze heel makkelijk de meldingen door kunnen geven. Anders dan doen ze het niet en dan zit er geen toegevoegde waarde in. Maar een ander belang is dan, dat je zegt, het moet wel heel veilig zijn, die interface en die connectie. Maar, hij moet wel performen, want ik bedoel er zijn een aantal van die systemen, en het gaat wel om zulke hoeveelheden per uur tijdens piekmomenten. En dan moet ik ook nog in staat zijn om die SECUNITE nummers die ik krijg, te koppelen aan die beveiligerspasjes (BIDs). Ergens moet er hier een koppeling plaatsvinden. Dan hebben we nog een concern: ik laat niet al mn beveiligers door dat registratie proces lopen. Dat willen ze niet. Dus zeggen ze van we hebben hier een SECUNITE HR systeem, en vanuit daar pompen we wel de data naar jullie toe. als er hier in dit HR systeem wat wijzigt dan moet dat hier in het registratiesysteem ook wijzigen. Dit is de context, we moeten hier een klein deel van scopen.

Er kan een belang zitten tussen het op een gemakkelijke en veilige manier koppelen, en de performance die het systeem moet leveren. Er kan natuurlijk ook -

Dit systeem (in-uitcheck) is natuurlijk cruciaal, dan kunnen de meldingen niet doorgaan. Dat SECUNITE systeem werkt wel, je gaat er vanuit dat dat gewoon werkt. Sterker nog, dat ligt buiten onze verantwoordelijkheid. Er zijn ook werkplaatsen waar geen SECUNITE systeem staat, daar moeten we rechtstreeks interfacen met dit systeem. De branchevereniging beveiligers Nederland heeft gezegd, dit systeem moet echt 99,99% beschikbaar zijn.

Op welk level definieer je nou een concern? Is een perspectief altijd een QA?

Hier is een heel duidelijk concern, is er een systeem veilig? Een ander concern is hier van is de privacy van de werknemer geborgd? Hangt natuurlijk nauw samen met security. *Om het proces van erover nadenken te starten zou ik misschien wel wat kunnen vragen als input.* ***Initial fill methode***

De valkuil is wel dat met name alles belangrijk is. Ik zit nog even te twijfelen over concerns en perspectieven. Wat is nou de relatie tussen de twee. Uitleg perspectieven: die zijn om te categoriseren en die concerns juist bij elkaar te houden. Zodat je twee perspectieven, of 1 en het zelfde perspectief tegen elkaar kunt uitzetten, zonder dat je de draad kwijt raakt.

Bijvoorbeeld, ik moet 99,99% available zijn, die zou je in een resilliance of een performance perspectief kunnen plaatsen. Een ander voorbeeld kan zijn je response tijd: hoe hanteer ik nou dat het inchecken zo werkt, dat een beveiliger geen 10 seconden moet wachten tot het systeem zegt; Oke, jij mag naar binnen. Dat zou ook een concern kunnen zijn voor de scope: registratie-aanmeld systeem en dan op performance perspectief. Wat ik voor me zie in de methode is dat we dan ,stel we nemen performance en security, er zijn concerns die onder deze perspectieven vallen en samen botsen. Performance zou zo maar een perspectief kunnen zijn. De insteek is dat je vanuit een bepaald perspectief kijkt naar je architectuur of de beoogde architectuur. En dat je binnen of via zo'n perspectief juist op zoek gaat naar bepaalde concerns die dus van belang zijn. Uiteindelijk gaat het om het vinden van die inconsistenties, en wil je daarom de bepaalde groepen tegen elkaar uitzetten.

Financieen is natuurlijk altijd heel makkelijk, het mag vooral niet te duur zijn. Maar je zou kunnen zeggen, hier heb je natuurlijk performance en beschikbaarheid die een perspectief vormen, en als je dat

gaat uitzetten tegen financieen, zeg je: we hebben deze oplossing bedacht, en die gaat uiteindelijk draaien op een stuk hardware. Op het moment dat die hardware niet voldoende blijkt, dan moeten we hardware bijkopen. Alleen dat moeten we zelf betalen. Mijn concern is dat ik het niet runnend krijg binnen het hardwarebudget. *en die botst misschien wel weer met bijvoorbeeld responsetijd? als ik een maximum responsetijd heb van 1 seconde, heb ik juist een enorm rack van servers nodig.* Zullen we eens even kijken hoe ver we komen?

Dan nemen we performance als groep. Availability is natuurijk geen performance concern, maar meer richting resilliance? Wat bijvoorbeeld een concern is in dit voorbeeld is de piekbelasting. Kan ik die piekbelasting opvangen? Dat is een extreme piekbelasting. Een ander concern is - deze twee systemen zijn gekoppeld - hoe kan ik waarborgen dat de performance van het portaal hoog blijft? Beveiligers moeten hun beveiligingspasje kunnen blijven aanvragen. Dus eigenlijk die pijl onderin die de twee databases met elkaar verbindt, die moet super solide zijn. Dus ik heb een performance concern dat mijn portaalfunctie dat al mijn schermen binnen drie seconde blijven draaien. Al heel snel is dat je NFRs aan het ophoesten bent. Als zorg ik ervoor dat mijn portaal niet te traag reageert wanneer mensen zich aan- en afmelden. Het portaal (=-niet aan-en afmeld systeem) moet echt stabiel blijven tijdens de piek. *Het opent wel de discussie.*

Vanuit security oogpunt zijn er ook een aantal concerns te definieren. Ik mag alleen maar aan- en afmeldingen behandelen van systemen die ik ken, en die ik vertrouw. Ik heb een concern dat de communicatie van ons systeem en het systeem van de SECUNITE, dat iemand daar tussen kan zitten. Ik heb twee concerns, dat de data die ik opsla, niet opstraat komt te liggen. Een ander concern dat ik heb is dat ik zeker weet dat mijn data van een aan en afmelding ook daadwerkelijk van een aan-of afmelding komt. Mijn concern: geen frauduleuze aan of afmeldingen in het systeem.

Bespreken matrix
Volgens de methode kijken we nu of er ergens een conflict of overlap is. Dan moet je weten hoe elk concern in de architectuur is verwerkt. Daarvoor hebben we dus die design rationale aangelegd. Hoe zou je het concern 'beveiligen van data' in de architectuur terugkomen. Ik ga gewoon die database encrypten. Want als er dan iets lekt, kan niemand er wat mee. Dit heeft directe impact op performance. Het duurt langer, omdat je alles moet encrypten. Als de beslissing dus genomen zou worden om alles te encrypten? is de encryptie van een database ook van invloed op de responsetijd of stabiliteit van het portaal aan de andere kant? **Participant wil ook performance tegen performance uitzetten. Grootste limitatie aan de matrix is wel dat hij 2D is.** Beschermen uitlekken data betekent wel dat de toegang tot het portaal volledig geautoriseerd zijn. Maak ik mij druk om performance, nee.

Geen frauduleuze aan- en afmeldingen. De oplossing is makkelijk. Ik vertrouw dit systeem (SECUNITE poort systeem). Dat is uitgangspunt 1. Ik accepteer alleen meldingen van systemen wat ik vertrouw. Dus wat ik doe, ik geef hem een certificaat, en met dat certificaat moet hij zn aanmeldingen doen, en zolang hij dat certificaat bij zich zelf houdt, dan krijg ik gecertificeerde meldingen binnen. Dus op het moment dat er iemand is die een melding doet zonder een certificaat dat ik ken, weiger ik die gewoon. Dat betekent wel, dat ik voor elke melding, moet ik het certificaat checken, en dat zou wel eens een dure operatie kunnen zijn. Elke individuele melding die vanuit SECUNITE komt, daarvan moet het certificaat gecheckt worden. Als ie vanuit SECUNITE komt dan ga je er vanuit dat het ook echt een werknemer is met een ID. Voor andere systemen publiceer je gewoon geen interfaces dus die kunnen er niet op aansluiten. Dat is duidelijk.

Is dat in conflict met performance - piekbelasting? Ja, want dit betekent gewoon dat het aan en afmelden meer tijd gaat kosten, of in ieder geval kostbaarder wordt. Dus dat is een conflict. 'Geen frauduleuze aanmeldingen' op de stabiliteit van het portaal. Geen conflict.

We gaan nu een beslissing maken, want we hebben iets tegenstrijdigs gevonden. De concern: geen frauduleuze aanmeldpogingen is in de architectuur aangepakt door certificaten en certificaaatchecks te doen. Dat staat haaks op de concern opvangen piekbelasting. Dat is lastig. De oplossing is simpel, ik accepteer elke aan- en afmelding, en die registreer ik, en de controle of ik dat systeem ken, of dat certificaat geldig is, dat doe ik pas later. En pas later wordt er gekeken, en gooi ik de ongeldige weg. De oplossing is dat ik een stukje asynchroniteit in het proces breng. De inconsistentie is dus dat je met een certificatensysteem gaat werken, maar dat dat ervoor zorgt dat de performance heel erg achteruit gaat tijdens de piekuren, daar waar je concern is om de response tijd zo laag mogelijk te houden. Dit kan niet als er certificaten worden gecheckt, tenzij je extreem buiten je hardware budget gaat. Deze inconsistentie heeft (als certificaten worden geimplementeerd), een hoge impact, namelijk de ochtend daarop staan er gigantische rijen voor de incheckapparatuur. De busineess value is ook hoog, omdat 1 van de 2 concerns is dat er geen frauduleuze inchecks gedaan mogen worden. Dus dan zou je dus de business value is hoog, en de impact is ook hoog. Op basis daarvan besluit je om hem te resolven.

Het interessante is, dit is een manier om hem op te lossen. Een andere manier om dit conflict op te lossen is dat je gewoon zegt, ijzer erbij. Gewoon paralelizeren die handel, en zorgen dat de database zwaar genoeg is, dan gaat het goed he. Op het moment dat je die oplossing kiest, dan zeg je eigenlijk, dat heeft weer impact op financiën.

Het zou dus ook leuk zijn als de methode de mogelijkheid bood - wat nou als je gekozen hebt om te resolven, hoe staat de oplossing dan weer haaks op de andere concerns? De output van de methode zou eigenlijk weer input voor de matrix moeten zijn.

Als ik dit ga bekijken dan zie ik een aantal oplossingsrichtingen elke oplossing moet ik wel weer toetsen tegen dit soort concerns. In de matrix specificeer je niet alleen dat je wat hebt gedaan, maar ook wat je hebt gedaan. Dus als je bijvoorbeeld zegt: ik postpone, ma**ar je geeft wel alvast 2 mogelijke oplossingsrichtingen aan in de matrix**, dan kan je dat later weer bekijken.

Tijdens je handling decision besluit je welk besluit je daadwerkelijk hebt genomen. **Maar al eerder inde methode kan je over mogelijke oplossingen nadenken. De participant denkt eigenlijk in die matrix al in oplossingen. Dus niet elke cel hoeft gecheckt te worden, je kan meteen activiteit 5,6,7 doen. De laatste drie stappen zijn iteratief.**
Nog 1 ding opmerken: dat met dat ijzer, dat heeft invloed op financiën. Als je gaat nadenken over het conflict. Dan weet je ook, hier moet ik nog het financiën aspect bij gaan trekken. **Bekijk meteen wat conflicten betekenen voor andere concerns. En bekijk ook wat mogelijke oplossingen betekenen voor andere concerns.**

Statements:

1. De stappen zijn heel duidelijk aangegeven, ik kan niet anders zeggen. De methode, of de richtlijnen: wat gebeurt er nou in die cellen ,wat ga je nu daadwerkelijk doen? Dat vind ik nog wat vager. De stappen zijn OK, de richtlijnen zijn NOK. Vandaar het cijfer 3.
2. Het is duidelijk wanneer de methode kan worden toegepast worden. Dit kan je doen in een offertefase, maar ook in een solution shaping fase. Op het moment dat de oplossing staat, en er komen nieuwe requirements binnen, dan kan je hem ook gebruiken. Of er poppen nieuwe concerns op. Eigenlijk kun je zeggen, de methode kan continu worden toegepast, het is een continu proces. Dat is voor mij helemaal duidelijk.
3. Technieken zijn beschikbaar, elke architect kan dit.
4. Kan ik deze vraag splitsen? Wenselijk staat soms tegenover haalbaar. Ik ben het ermee oneens, je moet echt altijd je stakeholders betrekken, liefst zo vroeg mogelijk. Dat is gewoon een 1. Haalbaar is echt context afhankelijk.
5. -

6. -
7. Dit is feitelijk wat ik doe, een van de aspecten van het werk van de architect. Je formaliseert het nu en dwingt de architect om hier bewust mee om te gaan, nu doe je het onbewust. Dat heeft natuurlijk in mijn hoofd gespeeld, met het shapen van een oplossing.
8. Daar ben ik het niet mee eens, dat is een twee (vier).
9. Moeilijk? Er zitten wat onduidelijkheden in de methode. Tijdens het gebruik merkte je al die discussie, van wat is nou een concern, en wat is nou een perspectief, 9/10 keer gaat het om nonfunctionals. En ik ben toch wel overtuigd dat de architect voor zichzelf moet bepalen met zijn eigen betrokken stakeholders en zijn eigen context moet afspreken van wat zien wij als perspectief en wat zien wij als concern? Zodat je in ieder geval een duidelijke rangorde hebt, daar zijn die perspectieven eigenlijk voor. Of moet ik dat zelf vastleggen. Die vrijheid moet erin, dat ben ik met je eens, daar moeten ze zelf over beslissen. Alleen mensen die de methode niet kennen, die moet je een handvat geven door ze een aantal hele goede voorbeelden te geven. Een scenario, met een geprioriteerde lijst van concerns, met een matrix (of aantal ingevulde matrixen). Initial fill. Vandaar dat ik methode is moeilijk op een drie zet.
10. De methode is ook erg flexibel.
11. Het bewust worden van inconsistenties, het bewust omgaan met inconsistenties daar is het effectief in. Het bewust nadenken, maar daarvoor moet je ze inderdaad ontdekt hebben, ik denk dat je in je hoofd heb je ze wel ontdekt. Maar ik denk dat je met deze methode omdat je zo tegen elkaar afzet dat je beter in staat bent om inconsistenties die je nog niet had ontdekt, om die in kaart te krijgen, en die dus wel te ontdekken. Ik denk dat ik het wel een effectieve manier vindt, dus daar zet ik een vier neer. *Als je ze al weet is het een effectieve manier om ze te bespreken, en vaak weet je ze al wel, dus vandaar je twijfels. Dat zal ik er ook bijzetten.*
12. Ja het is zeker effectief, maar daardoor ook een beperking, omdat hij 2D is. Elke beslissing die je neemt om een inconsistentie op te lossen, kan effect hebben, kan een conflict opleveren met een ander concern.

Eigenlijk zijn concerns ook oplossingen: je hebt een concern, die vraagt om oplossingen, vandaar ook de design rationale. je kijkt of de oplossingen met elkaar in conflict komen. Daar komt weer een oplossing uit, en die kan weer met andere oplossingen conflicteren. Het is heel simpel, de mens is heel erg 2D ingesteld, en een matrix om dingen af te zetten is een goede manier om dit soort dingen te ontdekken. Ik zou wel bij die matrix blijven. Ik zou wel een 4 geven.

<u>Losse vragen</u>

Vond je die diagnose stap nuttig en waarde toevoegend?

- ik vind het nuttig om te kijken naar impact, risico, busienss value, en engineering effort. Maar de invulling, die punten, heb ik niet als nuttig ervaren. Let wel: misschien dat het komt omdat we er nu snel doorheen gingen. Het zijn goede vragen die gesteld worden, maar dan vervolgens het meten, dat is echt arbitrair. Als architect moet je een antwoord hebben op die vragen, en het is ook goed om die antwoorden vast te leggen, en die te communiceren aan de stakeholders. “De vragen moeten echt blijven hoor”.

Vond je de alternatieven die ik bood, die oplossingsrichtingen, vond je die nuttig en waardetoevoegend? Komt het overeen met wat je normaal gesproken doet?

- Je lost het op of je .. op het moment dat je hem negeert, dan was eigenlijk mijn uitspraak er was geen inconsistentie. In mijn ogen zijn er gewoon maar drie opties: of je lost het op, of je stelt het uit, of er is een workaround. (Negeren = er is geen inconsistentie, improve = zelfde als negeren). Op het moment dat je zegt, een inconsistentie is er, maar we doen er niks mee dan was het of geen inconsistentie, of er is sprake van een postpone.
- *De theorie erachter is dat als je alles in kaart hebt gebracht kan je beslissen of je iets wel of niet bijhoudt. Dan kan je dat in ieder geval bewust doen, zonder dat je er later als verrassing achter komt. Dan weet je wel, er zit iets fout.*

Vond je die factoren die ik je in het begin liet zien, denk je dat dat nuttig is voor iemand die de methode gaat toepassen?

- Ik denk dat het idee goed is. Of dit de juiste factoren aspecten zijn, dat is nu te kort dag om daar wat over te roepen. Kwaliteit van de factoren is een vraagteken? De factoren (mits goed) zijn wel behulpzaam.

Zou je de methode wellicht een keer willen gebruiken? heb je de intentie?

- Ja.

Zou je andere een andere context kunnen verzinnen om die matrix te gebruiken? In welke andere context is dit nuttig?

- We hadden al een hele discussie om tot de juiste concerns te komen, en de stakeholder mist die discussie. Dus die gaat onmiddelijk vragen stellen. **Het explicitiet maken van de concenrs zorgt eigenlijk voor vragen.** Wat het wel altijd goed doet is, naar bepaalde stakeholders toe, is dat je met een matrix aantoont ik heb redelijk systematisch naar de problematiek gekeken, en het gene wat ik nu met je ga bespreken, dat komt niet uit de lucht vallen.
- het zou ook nuttig kunnen zijn, als je met een stakeholder in gesprek gaat, en je ziet meteen waar je het over moet hebben. Normaal gesproken pak ik mn evernote erbij. Als je dan ook nog eens kan doorklikken naar het vakje waar je het over hebt, en je kan daar notities bij maken, dan is dat echt super! denk ik. Als een leidraad voor een discussie. Maar dan moet je het wel als een eigen instrument beschouwen. Als ik in gesprek ga, dan heb ik een aantal topics waar ik het over wil hebben. Gedurend het gesprek kijk ik of ik alles heb behandeld. Als leidraad is het superhandig. En de andere stakeholder mag dan best zien dat ik dat instrument gebruik, dat geeft hem alleen maar een goed gevoel.

Zou je tevreden zijn met de resultaten? Heb je meer inzicht in de architectuur? En heb je het gevoel dat je meer inconsistenties hebt kunnen benoemen?

- Ja ik denk het wel, ja, volgens mij wel. Wat namelijk goed is aan de methode, hij roept onmiddelijk discussie op. Het structureert gewoon je discussie. Dat is denk ik de belangrijkste kracht van de methode.

Wat zou je zien als belangrijkste obstakel?

- Op het moment dat je de concerns en de perspectieven op t verkeerde niveau hebt zitten, dan zou hij wel eens niet kunnen werken. Dus een obstakel zou kunnen zijn, hoe zorg je ervoor dat je die concerns op het goede niveau zijn. Deze uitspraak zou ik moeten toetsen. Het formuleren van concerns is lastig. *vooral omdat ze impliciet zijn.* Het expliciet maken van die concerns zorgt ervoor dat je in een bepaald hokje moet denken, terwijl je ze eigenlijk liever impliciet misschein wel redeneert over de concerns.

Heb je moeite gehad met de hoeveelheid informatie die naar je toe kwam tijdens de methode?

- Nee.

Het zou kunnen zijn dat de matrix te groot wordt als er teveel concerns in staan.

- Ja maar je prioriteert toch niet voor niets?

Zou je dingen anders willen zien?

- Wat zou ik veranderd willen hebben aan de methode?
- Wat in de methode denk ik verbeterd kan worden is de expliciete - wat verbeterd kan worden is van elke beslissing die je neemt, moet weer gecheckt worden. En volgens mij, op het moment dat je een beslissing neemt, dan kan je vervolgens vanuit de methode weer feedback geven op die beslissing. *Dus hoe zou je dat formuleren?* Op het moment dat er een beslissing genomen wordt, heeft hij een bepaalde impact. Het nemen van die beslissing, vereist, dat je verplicht wordt geadviseerd om terug te kijken naar je andere concerns. Er moet een stap in, om te kijken, wat is de invloed van deze beslissing op de andere concerns. *dus niet monitor impact, maar concreter maken en terugkijken naar je concerns (in de matrix, en ook in de longlist).* het kan namelijk best zijn dat een beslissing leidt tot nieuwe concerns. *Voorbeeld: asynchroon zorgt ervoor dat je niet meer real time bezig bent, en dat als iemand uitcheckt*

binnen het uur, dat hij dan niet geregistreerd staat, maar wel een uur gewerkt heeft. Als je in dat portaal ook inzag hebt in jouw aan- of afmeldingen, dat dat inaccuraat is, want daar zit dan een vertraging in. Is dat dan erg of is dat niet erg? Dat moet je monitoren.

Expert 2, CASE STUDY, 02-03-16 13:00 uur

Uitleg methode
2 disciplines, modulair opgebouwd, elke architect heeft waarschijnlijk een ander doel, je kunt kiezen welke practices je gaat doen. Eerste activiteit is planning, daarbij definieer je kort verschillende dingen die voor jou van belang zijn. Het is niet de planning van een development/architecture project, het is echt de planning voor het uitvoeren en bepalen van deze methode. Eerste activiteit is het bepalen van je doel. Er zijn grofweg 2 doelen te onderscheiden: ik heb al een bestaand product, en het tweede doel is: ik ga een nieuw product/architectuur ontwikkelen. Dat bepaalt ook in hoeverre je in de middelste stap gaat zoeken naar bestaande architecturele documentatie, of dat je meer conceptueel bezig bent.

Respondent is in verwarring: *je bedoelt hier niet de planning van het scrumritme of iets dergelijks, maar echt de planning van de methode zelf of niet? Je bent geen epic aan het bouwen? Dit loopt als het ware paralel?*

Het is echt bedoeld als losse methode, die naast of over je bestaande ontwikkelmethode ligt. Ik heb expres geen tijden of tijdvak aangehouden omdat ik niet teveel inbreuk wil maken op de vrijheid van een architect. De tweede stap die je gaat doen is het discussieren van de situationele methoden, situationele factoren. Zijn puur ter indicatie. Tijdens het kiezen van je scope doe je dit op functionele basis. Je kiest eigenlijk je onderwerp. Daarna betrekken stakeholders, op de hoogte stellen, ze vertellen dat ze betrokken gaan worden.

10:00 min Je ziet dus wel dat de architect verantwoordelijk is voor het kiezen van de scope en het bepalen van het stukje architectuur dat geanalyseerd gaat worden, en dat vervolgens de stakeholders gaat opleggen. Het is niet zo zeer dat je met stakeholders de discussie aangaat, over welke onderwerpen zouden we nou eens moeten bekijken. Omdat dat de methode meteen heel erg lang maakt, omdat je er dan nooit uitkomt. Deze methode dient echt een architect als geheugensteuntje. *Op gut feeling en op basis van inschatting ga je dus kijken wat interessant is.*

Dan kies je de perspectieven, en op basis van welk perspectief je kiest kies je ook de concerns. De methode gaat er wel vanuit dat de concerns al bekend zijn. De methode probeert niet de architect te beperken maar slechts aanvullend te zijn. Dan ga je ze prioriteren en de techniek laat ik vrij, maar er moet een shortlist uitkomen:

13:00 Heb je ook een advies, over hoe lang de shortlist minimaal of maximaal is? Vooral ook om het proces te helpen. Als ik een discussie heb met stakeholders of met andere architecten, kan het voorkomen dat dat heel breed wordt. Dan blijf je er maar dingen bijtrekken. Dan is dit heel belangrijk, ik snap dat er 130 concerns zijn, maar de methode blijft behapbaar als we nu focussen op max 5; welke zijn dat dan?

Matrix
Hoe spelen de design rationales hier een rol? Vraag over de matrix.
De matrix is een soort conceptueel model voor de architect om zijn gedachtes te ordenen. Uitleg wat er per cel gevraagd wordt. De cellen in de matrix geven als het ware meteen een to-do aan voor de architect en zijn team. De design rationale wordt er bijgepakt als je niet meer weet hoe het in de architectuur is geregeld. **20:00 min** Check architectural elements, dan check je ook diagrammen, modellen die gebruikt worden tijdens het ontwikkelen.
Dus als ik het goed heb: als je hier klaar bent, dan ben je met je betrokken stakeholders het erover eens, dit zijn onze concerns, en dit zijn de inconsistenties, hier clashen ze. Dit zijn onze inconsistenties.

Diagnose
Wat is de oorzaak van de inconsistentie. Die moet even kort geevalueerd worden, even kort nagegaan van hoe erg is de oorzaak. Als het een developer is die een klein foutje heeft gemaakt, dan is de oorzaak vrij oppervlakkig. Als het wezenlijke meningsverschillen zijn, dan moet daarover gepraat worden.
Identify cause is meestal een hele dunne lijn met conflict of interest. Als er een conflict is tussen verschillende aspecten/aannames/assumpties, dan impliceert dat eigenlijk al wel een soortement van cause.

22:30 Heel vaak zul je natuurlijk zien dat het impliciet is, heel vaak zul je hem bij 'conflict' al weten. Goed wel dat je hem noemt. Want inderdaad bij dit voorbeeld (data ownership vs rollback mech.) dat we nu pas zien nu we erover na gaan denken zie ik dat hier iets verkeerd is gegaan, maar dat zal heel vaak zijn doordat je dan zegt; shit we hebben hier cassandra geïntroduceerd. Dan weet je die cause eigenlijk al. Maar het is wel goed als ze als stappen te benoemen.

De volgende is eigenlijk classify. We hebben een inconsistentie gevonden, maar wat gaan we ermee doen? Iedereen heeft altijd overal belangen. Ik heb geprobeerd uit de interviews en de literatuur de belangrijkste 4 aspecten te pakken. En daar controlevragen bij te zetten. Op basis van de controlevragen krijg je een score.

Wat bedoel je met is there a negative impact? ***niet meteen duidelijk voor participant.*** *Participant vraagt naar de betekenis van de score.* Leg uit dat dat voor systematische bedrijven is, waarbij het dus wenselijk is dat er een score wordt opgeslagen voor latere inzage.

Handling
Uitleg handling, ik zie het als drie schuifstandjes, en er zijn meerdere standen te verzinnen, maar dit geeft alvast weer een indicatie. Uitleg alle individuele oplosopties: bv. business high, impact high, effort, low => meteen solven. impact low, business high, effort high => P2Stpone, uitstellen.

31:00 min. de resultaten van de cyclus gaan in een knowledge base, en dat kan een dropbox folder zijn, maar ook een tool zoals Topdesk ofzo, als je maar weet wat je ervan geleerd hebt, of je nieuwe inzichten hebt, welke inconsistenties je hebt gevonden en dat je deze bijhoudt, welke concerns voor problemen zorgden, en als je er maar voor zorgt dat je de volgende keer niet het zelfde hoeft te doen. Followup acties kunnen nog verderreikend zijn zoals bijvoorbeeld kennissessies etc.

Final comments: de methode ligt als het ware over de bestaande ontwikkelmethodiek heen, dus als je agile werkt, kan je hem incrementeel toepassen, tussen sprints door, kun je het zelf kiezen. Als je een wat meer watervalachtige methodiek aanhoudt dan zou je kunnen zeggen bij design pakken we concern prioritization en concern modeling, als we gaan bouwen dan pak ik inconsistency monitoring en die matrix, en bij het implementeren gaan we kijken hoe we die inconsistenties het best kunnen resolven. En dat is het wat meer een watervalmethode.

35:00 Case voorbeeld
Waar we mee bezig zijn is om op onze webshop de levertijden beter te kunnen berekenen. Dus beter aan een klant te kunnen vertellen dan komt jouw pakketje. Of andersom, beter de klant te kunnen laten kiezen, wanneer wil je hem hebben. Dat ging voorheen vrij grof, door te zeggen, deze artikelen zitten in die productgroep en deze in die, en voor 1 productgroep geldt deze levertijd, voor de ander een andere levertijd. Dat houden we opzich nog wel vast, maar we willen het dan voor 1 specifiek artikel kunnen bepalen, hoeveel voorraad er is, wat zijn je afmetingen, en met welke vervoerders kun je mee. Eigenlijk dat helemaal uit kunnen rekenen om vervolgens te kunnen zeggen voor een specifiek artikel: bij dit unieke artikel kun je ervoor kiezen om dit vanmiddag nog bij de albert heijn te krijgen. Voorheen hadden

we meer onzekerheid: als je in die productgroep zit, dan zal het wel kunnen. Dat gaat heel vaak goed, en de klant merkt het vaak niet eens. Maar omdat wij zulke grote aantallen hebben, beginnen die honderdsten, of duizendsten wel op te vallen, dus zeg maar 0,001 procent. We vinden nu zelf dat we dat moeten gaan verbeteren. Het ging soms nog wel eens mis, vooral rond de feestdagen dat we het net 1 dag verkeerd hadden. Bijvoorbeeld op stakingsdagen is dat ook wel vervelend. Als P2StNL staakt, dat is vervelend voor onze klanten. Nu willen we de klant dan specifiek in kunnen lichten. Of problematischer: de eerste dag dat er sneeuw valt, waardoor sommige routes niet meer gereden kunnen worden.

In ons landschap hebben we de DLIOP = delivery service, die gaan we helemaal verantwoordelijk maken voor het berekenen van alle levertijden, waar eerst ieder systeempje dat voor zichzelf deed. Vanuit P2S = offersysteem daar serveren we de klanten onze offers in, wat wordt geleverd voor welke prijs, die ging dat ook zelf bepalen. DLIOP wordt het hart van onze rekenmachine, als ik dit artikel erin stop, wanneer kan die dan bij de klant zjin. Nee, niet zomaar de klant, maar een specifieke klant. Dat is iets anders dan een Nederlandse klant uit Amsterdam. Als twee klanten op hetzelfde moment een product in hun mandje doen, dan kan je verschillende levertijden hebben, op basis van je locatie. Wat je hier dus ziet is dat er al best veel afwegingen zitten.

38:45 Voorheen konden we zeggen (performance), als een artikel in deze productgroep valt, dan is hij er morgen. Dat kan ik hier in WPS (webshop) in isolatie oplossen. Ik hoef alleen maar te weten in welke productgroep dit artikel heeft. Terwijl nu, moet ik het elke keer gaan vragen aan DLIOP. Iedere request, dat zijn er 300 per seconde op dit moment, in rustige tijden, dat gaat gewoon niet. Dan introduceren we een component ertussen, die toch wat gaat opslaan. Dan rekenen we 1x in het uur alles door, veel minder dan dat je nu alles in 1x op slaat, en toch aardig ‘realtime’. Daar zit je op een afweging, als je hem echt realtime zou willen, dan moet je gewoon heel veel ijzer inzetten. Ten opzichte van beheerbaarheid en schaalbaarheid moet je dat gewoon niet doen. Kosten ten opzichte van performance, dan zoeken we een oplossing ertussen. Daar zit een afweging die we accepteren en daar werken we eigenlijk omheen door een component tussen te bouwen. Op het functionele gebied levertijd, dus dan hebben we de scope, pak ik de twee perspectieven performance, en informatie.

Dan hebben we dus concerns die erbij komen kijken, en 1 van die concerns is dus dat je realtime informatievoorziening wil hebben. Dat is echt een concern. Dat is altijd een afweging met performance. De concern is of het realtime kan bij informatie perspectief. Bij performance zou ik zeggen, de hoeveelheid servers. Een ander concerns is correctheid/accuraatheid. Responsetijd, is een belangrijke concern van performance. Resilliance is een belangrijke concern die ik ook heb als ik naar performance kijk.

Zijn er op dit moment concerns die elkaar niet/wel overlappen? Vrijwel allemaal op de eerste rij. Op de tweede rij wat minder. Dus realtime inzage levertijd met responsetijd en met resilliance. En correctheid met responsetijd overlap maar geen conflict, en resilliacne met correctheid ook overlap maar geen conflict. Al is responsetijd weer wel in conflict met correctheid, omdat als je te traag reageert, dan is je voorraad misschien niet beschikbaar, en dan ben je dus niet meer correct.

Ik denk dat je de verschillen tussen conflict en overlap in de praktijk heel gauw als 1 stap gaan doen, en ik denk dat het wel echt goed van je is om die dingen op papier te scheiden, maar in de praktijk komt het er wel op neer dat je gewoon 1 stap doet. *Voor een workshop is het altijd fijn, ik bemoei me nog niet met de keuzes die er allemaal onder zitten, ik ga ff heel snel bepalen wat we moeten bediscussieren, dan is dit een prima stap.* Dan zeg je eigenlijk van oke, we moeten naar deze kijken. Deze verdient gewoon aandacht. Qua stappen snap ik ze wel hoor dat je ze opschrijft. maar of je ze ook altijd zo uitvoert dat is in de praktijk wel een andere vraag denk ik.

49:00 Nu ik het zo zie, is servers en response tijd niet het zelfde? Servers zijn gewoon geld, he, je moet die dingen kopen. Servers zijn een middel en repsonsetijd is het doel dat je wilt halen. Want wat doe je voor die responsetijd. Dan is het wel gek dat ik hier zeg 'at ease' en dat ik hier zeg niet. *Participant komt tot inzicht in eigen keuzes.* Dat komt omdat servers onder responsetijd zitten. Een server is een beetje een afgeleide van responsetijd. Responsetijd vul je in met servers. (servers = design decision) Dat is toch wel een goede hoor, voor jezelf ook, je zou eens moeten nadenken over wat voor dingen je op die assen wil hebben. De vraag is of middelen ooit zinvol zijn op die as. Of dat je je middelen altijd wil vertalen naar het doel dat je hebt. Dus ik zet servers in ten behoeve van responsetijd. **design rationale, keuzes user stories.** De ontwikkelsnelheid is een doel, en je ontwikkelcapaciteit zou bijvoorbeeld een middel zijn. In de methode moet zeker komen te zitten wat je kan verwachten op die assen. Het is een gevoel. *Wat voor concerns kies je eigenlijk om op die as te zetten?*

53:00 min Hoe highlevel moeten de concerns zijn, hoe concreet moeten ze zijn? Misschien moet je zelfs wel op de assen zetten: hoge performance, hoge responsetijd etc. Dus eigenlijk moet de architect zeggen, oke, we hebben de shortlist gemaakt, als we specificeren gaan we meteen aangeven van oke, kijkend naar onze scope en ons doel, wat zijn dan de concerns de requirements die eraan hangen. Hoe specifiek moet een concern zijn. *Prioriteren kun je doen met een grote groep, evenals de matrix, maar concern modeling moet je even als architect alleen doen.* Als je er 20 man laat roepen krijg je allerlei dingen. Dan kan je gewoon de architect een vorstel laten doen, van zo moet het worden, en ik heb de concerns zo verpakt. De reden hiervoor is dat het lastige dingen zijn om te kwantificeren. Daar kun je uren over discussieren, dat is nooit goed of fout. Daar moet je op een gegeven moment gewoon een keuze in maken. Zet ze tegen elkaar af, als we ze nu niet meenemen dan nemen we ze de volgende cyclus wel mee.

Als we naar de matrix kijken zoals die er nu voorstaat, zijn dit vooral mogelijke inconsistenties. Want we hebben dit nog niet gebouwd. Dus dit zijn dingen die we nog op moeten pakken in het verhaal. We hebben nu eigenlijk het tweede doel gedaan. Omdat onze situatie zegt dat we nog geen architectuur hebben. Dus dan is het vooral proberen op voorhand deze concerns te benoemen en te kijken welke kant gaan we op. En dan ga je een richting kiezen. Wat mij betreft is het juist super goed om het zo op 2 (discovery en management) manieren aan te snijden, want deze stap (diagnose/handling) ga je alsnog doen in beide gevallen, en wat ga ik daar dan mee doen.

55:30 Als we naar die kijken, die cause, dan zie je repsonsetijd tov realtime levertijd, qua informatievoorziening wil ik hem realtime hebben en realtime is altijd de vraag, wat is realtime? Weet je, als je wil laten zien dat je pakketje naar huis komt, dat de P2Stbode onderweg is, dan is 5 minuten ook misschien wel voldoende realtime. Terwijl voorraad beweging kan van seconde tot seconde veranderen. Realtime, daar zit altijd caching tussen, en caching is prima, maar niet als het van seconde tot seconde verandert. Daar zit een tradeoff in: wat moet onze response tijd zijn, om toch nog voldoende mate van realtime van die levertijd te hebben. Wat is nou de minimale eis voor de maximale responsetijd?

56:30 Eigenlijk vertaalt zich dat (responsetijd) hier naartoe, naar het oplossen van een informatievraagstuk, en eigenlijk is dat ook een business vraagstuk, dus hoe realtime moet die inzage in levertijd zijn? En dan is stock toch ook al bijna realtime. We hebben dus gezegd wat we helemaal kunnen doen is voorberekenen wat voor datums mogelijk zijn, onder de aanname dat er voorraad zou zijn, de enige check die we hier op het laatste moment nog doen is even checken hoeveel voorraad er echt is. Want ik heb al bekend gemaakt welke leverdatums mogelijk zijn, dat weet de SLI (bestelling offer integrator). De webshop bevraagt de SLI. Laten we datums zoveel mogelijk voorberekenen, en op het allerlaatste moment de voorraad ertegen aan plakken. Je kan natuurlijk wel zeggen, als er voorraad is kan ik morgen leveren, is hij er niet dan zou eht volgende week maandag pas kunnen.

Dus de tradeoff zit hem dan ook in 100 procent kunnen we hem niet oplossen, realtime. Want zelfs daar zit iets vertraging. En dus een beetje een soort workaround. We accepteren de inconsistency, maar we moeten hem oplossen, dus een soort workaround, dus dan gaan we maar 2 sporen inzetten. Ene = een stuk voorberekenen, en het ander er realtime tegenaan plakken. Dat is een beetje de uitkomst van deze discussie hierzo. Dus stel dat dit gebouwd was geweest, zou dit dan een discussie zijn? Dan zouden we erachter komen dat de DLIOP veel te veel load ging krijgen. De eerste insteek was responsetijd realtime inzage informatie, ik ga hem dus continu en realtime aanroepen. De inconsistentie uit zich dan door de load op de DLIOP, en om dat op te lossen zou je dan heel veel servers bij moeten bouwen. Dat lukt gewoon niet.

59:30 Stel dat dit dus was gebouwd, dan is er een inconsistentie: ik wil aan de ene kant wel die realtime levertijd maar mn responsetijd gaat gewoon gigantisch laag omdat ik er uiteindelijk te weinig rekenkracht achter heb zitten.

De volgende is ook interessant: eigenlijk, de webshop, als er een klant komt, die moet het gewoon altijd doen. Als er een service omvalt ergens moet de webshop het nog gewoon doen. Daar hebben we wel een uitdaging. Stel dat de webshop het nog wel doet, maar hij (DLIOP) valt uit, dan heb ik geen realtime inzage in levertijd meer. Daar hebben we eigenlijk hetzelfde, oke, ik wil de juiste inzage in levertijd hebben, maar ik kan niet anders, we moeten er vanuitgaan dat hij er een keer niet is, dus daar moet ik ressilient op zijn. Als ie er dan een x niet is dan kan ik geen inzicht in levertijd geven. Dat lijkt een beetje paradoxaal natuurlijk: je wil realtime inzicht, daarom bevraag je de DLIOP, maar als hij er dan niet is, dan kan je geen inzicht in levertijden geven. (normaal kan dat wel als je gewoon asycnhrone en niet realtime informatie stuurt, die verandert niet en dus vrij statisch, als er dan een service uitligt zijn er geen actieve checks nodig, vandaar dat het paradoxaal lijkt om aan de ene kant realtime te willen gaan, en aan de andere kant ook super resilliant te zijn: wat je bent als je asynchrone communicatie gebruikt). Dat is een conflict. *Is het ook een inconsistency?*

We lossen dit op door te zeggen: als de DLIOP uitvalt, doe dan gewoon alsof alle voorraad aanwezig is voor de levertijd. Vertrouw dan volledig op je P2S. Als hij dan weer gefixt is kan je weer inzage krijgen, dit gaat echt maar om een paar seconden. In die paar minuten dat hij out is, heb je dan wel weer zoveel 300*60 300 per seconde x 60 seconden zoveel klanten teleurgesteld die een artikel aan het bekijken zijn. Dat is gewoon ruk. Dat wil je echt beperken. Nou, een aanname die kan is laten we dan maar gewoon de eerstmogelijke datum serveren (initieel) en dan checken en later zeggen, helaas sorry, is toch een paar dagen later.

Correctheid en realtime inzage liggen eigenlijk heel dicht bij elkaar. *Participant beseft zich dat nu ineens.* Responsetijd moet altijd onder het maximum liggen. Realtime is ik wil nu een antwoord, maar dan moet het wel correct zijn. Je kunt natuurlijk een antwoord geven dat niet correct is, wat wel dom zou zijn.

Voor de eerste hoog, de impact en risico, geen juiste inzage betekent gewoon dat je verkeerde beloftes gaat doen aan de klanten. Daarmee is de business value ook hoog, business value is vaak ook een beetje impact. Engineering effort is ook erg hoor, om dit op te lossen. Ik denk dat ze alle dire hoog zijn voor deze cel, en dat zou er toe leiden dat je zegt we gaan deze aanpakken: *ja dat gaan we ook doen, dus in die zin sluit dit goed aan op de stappen Diagnosis en Handling.* Sterker nog, we gaan niet eens live zonder.

Uitleg tool 1:09

1. Ik denk een 4. Wat het duidelijk maakt is dat je zegt van wie is er nou wanneer betrokken. Welke stappen zouden de stakeholders meer of minder betrokken moeten zijn. Een soort RACI achtig iets, denk daar eens over na. Responsible, Accountable, etc.
2. Fase gewoon duidelijk, geencommentaar.
3. Ik heb zeker technieken beschikbaar om de methode uit te voeren, tenminste de stappen waarbij dat nodig is. In die zin ik zou nog wel wat meer houvast kunnen gebruiken als ik hem uit zou moeten leggen de methode, wat is nou de definitie van een concern, wat is nou de definitie van een perspectief, een conflict. Even goed helder, wat zijn nou die dingen. Wat kan er nou wel en niet in die matrix?
4. Zo veel en zo vroeg mogelijk. Anders krijg je een ivoren toren verhaal.
5. Design rationale -/+, doe je vaak impliciet, en is vaak al aanwezig op het mometn dat je het gesprek ingaat. Of dat dus nuttig is weet ik niet. Misschien bij anderen wel. Kennisbank wel.
6. De methode is in mijn situatie misschien net wat minder handig, omdat deze discussies toch al vaak aanwezig zijn, en daar doen we het al op voorhand, wel impliciet. Ik denk dat de methode zeker kan helpen bij het expliciet maken wat we eigenlijk impliciet al doen. waar het voor mij wat lastig is, om helderder te krijgen wat de verschillende onderdelen zijn. Ik denk dat een aantal elementen hieruit enorm zouden kunnen helpen, ik denk dat ik met collega's heel snel die matrix in springen, laten we dit is tegen elkaar afzetten, (oplossingen?) omdat we die concersn wel vaak al helamaal helder hebben, dan kijken we liever naar hoe ga je die oplossen. Focus op lossing! Dan kijken we liever naar hoe ga je die oplossen. Dat toolkit mechanisme is heel goed aan de methode. daarom de 4.
7. Disvovery -/+ omdat we solving gericht zijn, management daarentegen wel +
8. Mijn vrijheid als architect wordt een beetje beperkt door inconsistency discovery, en niet door management. Binnen onze context, doe je veel van de stappen toch, alleen nu maak je ze explicieter. dus dat zou denk ik alleen maar fijn zijn denk ik, een ondersteunende rol. Dus nee dat voelt niet beperkend. Maar ik denk dat dat ook erg per bedrijf verschilt. Bij mijn vorige week had ik misschien wel juist dat bovenste stuk willen doen.
9. De methode is moeilijk; nee, maar even kijken naar de definities.
10. Flexibel, voor mijn gevoel wel. je kunt prima zeggen ik pak deze onderdelen wel en deze onderdelen niet.
11. Concerns heb ik een beetje mn twijfels over. Ik moest goed bedenken hoe ik de concerns ging noemen. Terwijl ik de afweging, waar ik het over wilde hebben, al vrij snel benoemd had. dus in mn hoofd had ikd e afweging wel duidelijk, het ging over het plaatje wat hier stond, maar toen moest ik even schakelen van hoe kan ik dat nou in concerns vatten. Ik weet ik wil het over dit vakje hebben, maar om dan tot dat vakje te komen, en dingen in 2 assen te plaatsen, vond ik even lastig. Wil niet zeggen of dat slecht is, maar ben er nog niet helemaal over uit of dat de meest effectieve manier is.
12. Het gebruik van een matrix is super. Als je de concerns eenmaal hebt, dan is het super effectief om ze in een matrix tegen elkaar af te zetten, er even doorheen te gaan, en afwegingen te maken. Misschien moet ik de definitie van concerns wel helemaal vrijlaten aan de bouwstenen waar jullie als architecten mee werken. Ik denk dat het goed is als je een soort template maakt, waarin een soort voorbeeldmatrix al gegeven is. Het zou wel helpen als de toolkit/methode een aantal perspectieven met concerns zou leveren. Heel vaak kom je wel op hetzelfde uit, maar noem je het net ff anders. op welke ordegrootte moet ik zitten? waar moet ik over nadenken? deze tool is opdeze as al een keer beproefd, dus zo zou ik het doen. En dan ben je als architect prima in staat om voor zijn organisatie dat aan te passen.

Diagnose stap:
Als de vragen wat specifieker zouden zijn voor onze situatie, als ik dat dan zou voorbereiden zeg maar, dan denk ik dat dat heel zinvol zou zijn om te beslissen wat je hiermee wilt doen. daarmee bedoel ik

dus ook echt organisatiespecifiek: dus de organisatie moet deze zelf specifiek maken, zodat ze ze vaker kun hergebruiken. Ik denk dat als we hierover beslissen volgens de diagnose stap, dat we niet per se de scores zouden aanhouden maar meer een indicatie krijgen van high medium low.

Handling stap:
Ja, ik denk dat het wel een goede is, want je ziet dat het structuur geeft aan het zoeken naar oplossingsrichtingen. Meestal - stel er is een issue - ga je daar met stakeholders over discussieren, terwijl het eignelijk wel goed is, stel we hebben 6-7 alternatieven, eigenlijk helpt het ons om dan te zeggen, stel we werken er omheen, wat doen we dan. Stel we gaan het ergens oplossen, we hebben heel snel de neiging om allerlei mogelijke oplossingen te gaan verkennen, maar meeestal zijn het verschillende invulingen van jouw zes punten. Het is wel een mooie kapstok om de discussie te voeren.

Situationale factoren: denk je dat die profielen van nut zijn inde methode?
Ja ik denk het wel. alleen ik denk in onze organisatie we sneller naar het onderste gedeelte van de methode zouden neigen, en niet zozeer naar de bovenste. Issues komen voorbij en dan pakken we ze op zegmaar. Ik denk dat wij niet echt zozeer opzoek zijn naar inconsistencies, want die hebben we al voldoende, alleen even iedere sprint bedenken welke willen we nu oppakken. Dat is meer de issue. Issues worden continu aan de backlog toegevoegd, en soms denken we als hij nu te groot is dan pakken we hem dan gewoon op. Dat prioriteren zit eigenlijk gewoon helemaal in ons proces.

Zou je de methode gebruiken? Ja. ik denk dat de aspecten uit de laatste drie blokken - het inzichtelijk maken dat er een inconsistentie is en het inzichtelijk maken van die inconsistentie, daar zijn wel best goed in, en dat hebben we al impliciet. Ik denk dat we dat al een beetje toepassen. Wat ik nou wel formeler wil doen, is zeggen van oke stel je hebt deze 6 opties om het op te lossen, dus die workaround, solven uitstellen, dat vind ik wel een mooie manier om het probleem te bespreken. Zou ik zeker ook wel willen hanteren.

Wat is het sterkste punt? Het is een methode die niet heel theoretisch is. Je hebt heel veel architectuurframeworks en methodes die heel theoretisch zijn. Dit is volgens mij heel praktisch en heel toepasbaar. Daarnaast levert de methode ook gewoon hele praktische resultaten, dus niet alleen we beschrijven iets of we brengen iets in kaart, maar we hebben ook gewoon een oplossing. Dat is voor mij het meest krachtige eraan denk ik.

Wat ik nog het lastige vind: de matrix vullen dat voelde soms een beetje gekunsteld aan op sommige vlakken. Dat komt omdat we niet heel lang stil hebben gestaan door wat onze concerns nou precies zijn, en door de tijd. Daar zit een beetje mijn zorg, hoe snel en gemakkelijk krijg je die assen goed tegen elkaar afgezet.

Heb je het gevoel dat je iets meer inzicht in je architectuur hebt gekregen? Ik denk als je hem goed in zou vullen, dat de knelpunten die je impliciet zou benoemen en meestal voor jou (als architect) bijzonder helder zijn en voor een directe collega misschien ook maar voor de rest iets minder, dat daar de methode vooral helpt om god uit te kunnen leggen waar is hij ontstaan, wat wegen we nou tegen elkaar af, daar helpt het zeker bij. Dus als communicatiemiddel, is het erg makkelijk. Daar is het wel erg zinvol voor.

Wat zou er veranderend? Template opstellen zodat je het makkelijk gevuld kan worden en architecten een voorbeeld hebben. Ook duidelijk wat precies concerns zijn.

H.3 Evaluation: Expert interviews

Expert 3, EXPERT INTERVIEW, 01-03-16, 18:00 uur

Uitleg methode:
Eerste vraag is eigenlijk, in wat voor fase zitten: hebben we al een bestaand product, of gaan we een nieuw product maken. Als je nog moet gaan beginnen, dan is het meer op conceptueel vlak, heb je al een product dan ga je zoeken in je systeem en documentatie. '***Dan is het dus een achterafbepaling zeg maar'.*** Dat bepaalt ook meteen de snelheid en de intensiteit waarmee je door de methode heenloopt.

Dan ga je naar de situationele factoren. Dat is eigenlijk een manier om te bepalen hoe je als bedrijf deze methode kunt doorlopen. En dat hangt dus af van een 14-tal factoren die ik heb geidenticieerd. De architect wordt wel aangeraden om zelf te kijken wat vind jij belangrijk. Voorbeelden zijn tijdsdruk, stabiliteit van de omgeving. Ander voorbeeld is teamgrootte. Hoe groter je teams, hoe systematischer je moet werken.

Zo ontstaan er drie profielen, en de architect beslist dus op basis van die profielen hoe die de rest van de stappen eruit komt te zien. Daarna is het de vraag hoe, waar en hoeveel stakeholders je wil gaan betrekken. Als je met vragen komt te zitten over inconsistenties, kun je stakeholders betrekken om conflicts te solven etcetera. Als je de methode wat systematischer uitvoert, adviseert de methode om grotere groepen stakeholders te betrekken.

> *Wat bedoel je precies met grote groepen stakeholders? Is dat de hoeveelheid mensen die je uitnodigt, of de diversiteit groot?*

Diversiteit. Dat legt hij vast in een klein project proposal/document. Afhankelijk van hoe snel die er doorheen gaat. Volgende fase, je kiest de perspectieven die belangrijk zijn om te evalueren naar aan leiding van je project goal. Van die perspectieven kies je dan concerns in de stap Gather Concerns, de concerns die op dat moment handig zijn. Daar geef ik een uitleg voor, maw je kan elke techniek kiezen die je wil, als je zelf maar een prioriteit hanteert. Als er maar rangorde ontstaat in de longlist.

> *Andere vraag: je zegt van je kiest een perspectief en vanuit een perspectief bepaal je welke concerns je gaat nemen.. Dan neem ik al aan dat al iemand die concerns gecategoriseerd heeft in bepaalde perspectieven.*

Dat moet ik er inderdaad bijzeggen, de methode neemt aan dat de concerns al geidentificeerd zijn. Deze methode probeert niet over te nemen wat er in de praktijk al gedaan wordt. Een vereiste is dat de architect al de concerns al heeft.

> *Dan moet je dus ook duidelijk hebben wat voor soort concern het is. Het kan zijn dat een technisch concern is, is mijn omgeving op tijd klaar. Het kan zijn dat het een functioneel concern is, van werkt dat wel met elkaar.*

Dat perspectief moet van te voren duidelijk zijn per concern.

> *Iemand of een instantie moet die concerns categoriseren, dat zal je in de regel doen met belangrijke stakeholders. Stakeholders zijn er natuurlijk op allerlei vlakken, ook een infrastructuur-afdeling is een stakeholder.*
>
> **Stakeholders kunnen daar al betrokken worden.**

Die concerns komen in een shortlist terecht, in de shortlist ga je de concerns specificeren, dat houdt in dat je eigenlijk begint met een concern alleen de term, maar in die stap ga je kijken voor welke stakeholders die concern nog meer relevant is, dus je hangt eigenlijk de stakeholder aan de concern.

> *Als je de stakeholders pas met de concerns associeert op het moment dat je die prioritering al hebt aangebracht, waarvan je niet eens zeker weet of je die wel vast kunt houden.*

Ik heb dit bewust zo gedaan, omdat als je alles eerst gaat specificeren, en je gaat daarna pas prioriteit eraan hangen, dan moet je een hele lange lijst door.

> *Wat verstandig is, is als je zegt, ik hou dit (concern prioritization) kort en ik zorg dat daar een lijstje uikomt wat ik ga gebruiken, je kunt niet van alles al gaan uitwerken want dan heb je een hele hoop werk, en dat levert misschien wel niks op. Wat je moet doen, is proberen op de belangrijkste eruit te pikken, die opgelost moeten worden, of die opgelost zouden moeten worden, waardoor je anders niet meer verder kunt. Bijvoorbeeld een infrastructureel probleem, dat moet opgelost zijn voordat je verder kunt. Je prioriteert het hier (prio) en hier ga je het uitwerken (modeling) maar dat is wel correct, want je kunt het hier (prio) ook nog niet volledig uitwerken want je wil niet andere concerns in gevaar brengen, je moet daar een balans in brengen, hoeveel heb ik hier nodig. Eigenlijk is dat een iteratief procesje.*

Daarna is het belangrijk, dat je een design rationale ontwikkelt. Als er al een design rationale is, is het belangrijk dat ie als input voor het proces geldt, of als hij er niet is, dat de architect en zijn team dat aanmaken. Voor de belangrijkste concerns, identificeer hoe die zijn verwerkt in de literatuur. Waar creëer je de link tussen je belangen en hoe ze zijn gemaakt in de architectuur. Een hele pragmatische architect doet dat misschien wel ad-hoc, of uit zijn hoofd, als je wat gestructureerder werkt ga je dat misschien wel documenteren.

> *Ook daar geldt natuurlijk weer als je in een groter bedrijf werkt is de diversiteit waarmee je te maken krijgt veel groter, als je in een klein bedrijf werkt zul je in de regel wel weten wat er speelt en wat er gespeeld heeft in het verleden, waar je nog mogelijk last van gaat krijgen. En dat moet je wel vastleggen hier, en naarmate de groep groter wordt wordt het belangrijker om dat goed vast te leggen.*

Presenteer matrix 10:30 min
Idee is heel simpel, er worden dus perspecieven gekozen, er worden daarna concerns ingeladen, en nu heb ik ervoor gekozen om mijn dataperspectief tegenover mijn dataperspectief te zetten. Concern tegenover concerns is een grijs vakje bij wijze van spreken. Maar je data ownership tegenover je rollback mechanisms, daar kan het bijvoorbeeld gaan wringen. En dan zie je dus dat het team op deze manier inzicht kan krijgen in waar die concerns overlappen. Het vullen van de matrix doe je in deze stap. De stap erna zegt eigenlijk van, is er een overlap, dus raken ze elkaar. De preconditie voor inconsistentie is dat er overlap is. De eerste stap is: kijk, en ga na of er overlap is tussen twee concerns. Want als er overlap is tussen twee concerns, dan is er ook overlap tussen twee diagrammen, (die die concerns addresseren). Dan zeg je, het is eigenlijk niet erg dat er overlap is, als er dan maar geen conflict optreedt.

De architect kan, met behulp van de matrix en de methode, met het team overleggen en dan besluiten wat voor status de cel krijgt. **(belangrijk om te vermelden dat hij niet elke cel hoeft te checken).** Bijvoorbeeld: at ease, of conflicting, of overlap. **belangrijk: inconsistencies moeten in de gaten gehouden blijven.** De matrix dient dus als platform om acties af te stemmen, of to-do's te maken.

13:00 min

Stel: we hebben een inconsistentie ontdekt: ofwel in de code/systeem, ofwel in een model. Inconsistenties in de code kan je gemakkelijk detecteren door middel van tools of tests. Deze methode bevat eigenlijk inconsistenties die niet met automatisch testen te vinden zijn. Laten we zeggen dat we hier dan een inconsistentie hebben gevonden. Dan kijken we eerst naar een oorzaak van een inconsistentie. Daarna ga je naar classificeren. Ik heb drie belangrijke aspecten gedefinieerd: impact, business value, engineering effort. De combinatie van deze drie bepaalt eigenlijk wat je daarna met een inconsistentie moet doen.

Uitleg classification model

Uitleg Handling (ap: handling decision tree maken!)
Eerste actie is wel, kunnen we hem oplossen. De methode adviseert uiteindelijk om hem op te lossen. De methode adviseert om de inconsistentie op te lossen als de business value en de impact hoog. Dan maakt het niet uit hoe veel het kost, want die 2 zijn zo belangrijk. Als de impact laag is, de business value hoog, maar de Required effort ook laag. Dan kan je hem ook oplossen. Ignoren doe je eigenlijk als impact en business value laag zijn. Je kan ook tolereren: uitstellen, (**volgend oplosmoment**), omzeilen, of verbeteren(commentaar erbij).

De vierde category bevat factoren die specifiek zijn voor elke inconsistency, zoals de cause, het type inconsistentie.

> *Ik zou trouwens die "will it happen again" verplaatsen naar 'impact'. Daar past hij veel beter thuis, en bovendien is dat heel belangrijk. Als iets eenmalig is, en heeft niet zoveel prioriteit, dan zou je er wellicht minder tijd in stoppen als dat het toch elke dag voorkomt.*

De laatste fase is finalization, je hebt dan als team besloten wat er gaat gebeuren, dan gaat er een oplossingsrichting uit of iets in de trant van een change request. Dan is het belangrijk om te monitoren gaat het goed, wat heb je ervan geleerd, wat kan er beter, wat zijn de dingen die het team kan meenemen? Dat gaat in de knowledge base. De knowledge base kan in dropbox.

> *Klopt, dit kan in elke soort tool, daar zijn er genoeg van. Wij gebruiken zelf Topdesk, om meldingen af te handelen. Maar ook klanten gebruiken die. Als jij een helpdesk belt, komt dat in topdesk te staan, dat wordt doorgestuurd naar een behandelaar. Alles wat je doet wordt vastgelegd, en als je in de toekomst weer zoiets hebt, kijk je eerst in topdesk heb ik al zoiets gehad of gedaan. In feite ook een soort van knowledge base.*

Daarna krijg je nog follow up questions, waarbij je ofwel nog een cyclus doorloopt van de CCIM methode, of wel andere viewpoints aanneemt. Of wel kennissessies organiseert om teamcommunicatie beter op orde te krijgen. Dan heb je eigenlijk de cyclus doorlopen, en dan kan je kiezen hoe je terug gaat via de pijlen in het schema. *Ik zou hier ook nog een pijl doen, van finalization naar concern prioritization. met Agile begin je ook met een soort grove prep fase waarbij je vaststelt hoe groot de sprints ongeveer worden en wat gaan we doen in die sprints zeg maar.*

Hoe lang zouden de stappen moeten zijn, en wat zou de duur van mijn methode worden?

> *Het is wel enigszins te vergelijken met een agile benadering van ontwikkeling, je probeert alles in stappen onder te brengen en je probeert te verdelen en een prioritering aan te brengen, en aan het eind van de rit ga je bekijken of je dat allemaal goed hebt gedaan, en of je het allemaal hebt opgelost. En als dat niet zo is, ga je weer terug naar het begin. En dat zal erg afhangen van het traject waarop je het gaat toepassen. Kijk als je een nieuw systeem gaat bouwen, nieuw polisadministratie, dat is in de regel een vrij ingewikkeld proces dat in de regel misschien wel*

> *een jaar of anderhalf kan duren, in doorloop tijd. Dan zou je dit hele proces, iteratief een aantal keren doorlopen, en en verschillende stappen meer of minder dan andere stappen. Dat hangt ook nog heel erg af van de complexiteit die je tegenkomt.*

Inderdaad, ik dacht zelf aan dingen als, heb je al iets op papier staan, hoe geoefend ben jij in je methode? De ervaring. *We zijn zelf bezig met de overstap naar Agile, dat kost in het begin veel tijd, omdat het een beetje onwennig is en iedereen heeft problemen met dingen aanleren enzo. Na een tijdje kun je veel sneller vooruit. In het begin zal het heel erg zoeken zijn naar wat heb ik dan nodig, wat voor documenten moet ik maken en hoe vul ik die matrix in en hoe ga ik daarmee om. En als je daaraan gewend bent, dan zal het veel sneller gaan. Of dat in uren of in dagen gaat, dat zou ik niet kunnen zeggen.*

Is het een beetje duidelijk, zo de methode?

> *Ja op zich wel, ik zie wel wat paralellen met de agile wereld en allerlei principes in het architectuur-werk, en dat maakt het toepasbaar omdat je de methode dan ook gewoon over bestaande ontwikkelmethoden heen kunt leggen en dat is denk ik wel belangrijk. Een compleet nieuwe methode aanleren, dat is veel lastiger, om te implementeren bij een bedrijf. Want een bedrijf heeft een bepaalde manier van werken, en hiermee kan jij op zich goed aansluiten op veel bestaande manieren van werken.*

Daar heb ik hem ook wel echt op ontworpen. *Bij sommige bedrijven zullen dit soort iteraties wel gewoon voorhanden zijn in enige vorm.*

presenteren tool
Dit geeft weinig detail, maar dat komt omdat we niveaus wilden behouden.
"Je wil het overzicht ook houden hier, he? Je wilt niet te veel detail want dan ben je het overzicht kwijt. Ik vind dat dat goed terugkomt maar ik zou wel willen dat als ik hier op klik dat ik een drilldown krijg van wat is er dan nou aan de hand".

33:30 min Wat vind jij belangrijk om in een inconsistentie in te zien? *Ik wil de prioriteiten inzien, want op basis daarvan bepaal je of je er wat mee gaat doen of niet, en wanneer je er wat mee moet. Eigenlijk gewoon de voorsteling van dit lijstje. Ik wil details hebben, ik moet een geheugensteuntje vast kunnen leggen. Ik wil weten waarom die die prioriteit heeft. Freeformat text veldje, daar kun je dan bv redenen inzetten waarom je een bepaalde concern naast je neerlegt. Er kunnen allerlei soorten redenen zijn waarom je even iets vast wilt leggen. Dat wil je terug kunnen zien. Dus je wil het overzicht hebben, maar op het moment dat ik een detail aanklik wil ik ook graag zien wat er met dat detail aan de hand is.* <Zie notes> *wellicht kun je een documentje of een URL er zelfs aanhangen, naar documenten of testverslagen.*

Dit screen is gemaakt voor als je de concerns hebt verzameld, hier kan je een concern invoeren. Je kan een naam geven, de betrokken stakeholders. Er is een dropdownlist om het perspectief te selecteren. Je kan belangrijke requirements invoeren, en een free text veldje voor comments. Dan klik je op opslaan, en dan komt de concern in de longlist terecht. Daar zie je wat voor concerns je hebt, welke er in de shortlist staan, welke in de longlist, etcetera.

Als je er dan op klikt, dan zie je de details van een concern die belangrijk zijn. Prioriteit, textveld, stakeholders die betrokken zijn. *Oke. Hier zie ik nog wel een interessante, 5 days left.. er kan natuurlijk een tijdsdruk aanzitten, dat moet je goed in beeld brengen.* **Invoegen in methode.** *Ze kunnen natuurlijk dringender worden naar mate ze langer op de shortlist staan.. Bij het prioriteren moet je daar al naar kijken. Daarom zei ik ook al: je moet terug naar prioriteren: je moet voortdurend blijven prioriteren.*

Vragen / statements (40:00 min)

1. *Ik ben het eens, moet er nog wel even goed naar kijken, iets wat een eerste keer uitgelegd wordt moet ik altijd doorheen.*
2. *Deze methode, - ik zou deze methode gebruiken het hele traject van aannemen tot het opleveren van het systeem zegmaar, het is een doorlopende actie, niet 1 specifieke fase binnen je proces, je blijft het voortdurend doen. Want je kan voortdurend tegen problemen aanlopen.*
3. *Wat bedoel je hiermee?* **Vraag niet duidelijk. Na uitleg wel.** *Elke architect of developer heeft deze technieken wel beschikbaar. Anders ben je geen architect.*
4. Sterk mee oneens. *Ik vind stakeholders erg belangrijk in elk proces, het is erg belangrijk om voldoende spreiding in je type stakeholders te hebben. Je hoeft ze niet allemaal aan te haken, maar ze zijn wel gedurende alle processen binnen je hele ontwikkeltraject erg belangrijk.*
5. Eens*. Topdesk verhaal.*
6. Neutraal. *Op dit moment niet zo, omdat ik nu juist meer onderhoud doe.*
7. Sterk mee oneens. *Methode voelt voor mij niet beperkend en dwingend aan. Een methode voelt voor mij nooit dwingend aan omdat ik de onderdelen pak die ik prettig vind en nodig vind. Een methode moet je ondersteunen, en dan heeft ie bepaalde voorwaarden waaraan je moet voldoen om met die methode te kunnen werken. En als je de methode ondersteunend vindt, dan neem je dat voor lief en dan beperkt het je niet en dan dwingt het je niet.* Als hij ondersteundend is, dan voelt hij niet meer dwingend aan omdat je de dingen eruit pakt die je nodig hebt. *Dat is een gevoel. Hij voelt voor mij niet beperkend of dwingend aan. Als ik die methode binnen mijn werk toe kan passen en ik heb daar baat bij dan is het prima. Als ik daar dan bepaalde stappen voor moet doen is dat prima: als ik mijn auto wil starten dan moet ik ook bepaalde stappen doen.*
8. Eens. *Ik denk dat het hooguit wat meer structuur toevoegt aan je proces. En dat is denk ik ook de bedoeling?*
9. *Nog niet gebruikt, daarom neutraal.*
10. *Wat ik tot nu toe gezien heb, met die losse onderdelen, lijkt het me zeker flexibel.*
11. *Met een concern bedenk je eigenlijk van als ik het ga bouwen, wat kan er dan misgaan. Een concern is eigenlijk waar moet ik over nadenken, wat moet ik oplossen, wat voor problemen kom ik tegen en hoe moet ik die aanpakken. En als je die problemen voor jezelf en de punten die je moet gaan oplossen en de requirements en alles op een rijtje gaat zetten, en je kunt die op een goede manier tegen elkaar afzetten, dan zul je zoals jij het nu geschetst hebt, dan zul je met die kruisverbanden sneller in staat zijn om erachter te komen waar je mogelijk op je plaat gaat. Een aantal dingen zullen elkaar niet in de weg zitten. Maar er zullen ongetwijfeld dingen zijn die elkaar heel erg in de weg gaan zitten, waarj e nogn iet zo gauw over had nagedacht. En als je dat gestructureerd kan weergeven, dat probeer je, dan kan het je helpen om die inconsistentie op te lossen.*
12. *Ook antwoord op vraag 12. Een matrix is een effectieve manier om overlap te ontdekken. Als jij goed in staat bent om de X en de Y as van die matrix in te vullen. En door je methode te doorlopen neem je concerns en prioriteer je die, en die ga je tegen elkaar afzetten. En dan kun je enerzijds bepalen hoe belangrijk een concern is en anderzijds hoe die tot problemen kan gaan leiden door die overlap vast te stellen.*

General commentaar op de statements: de vragen 3 en 8 waren een beetje onduidelijk.
Nog een paar losse vragen:

- Helpt de stap ‘diagnose’ met beslissen wat je moet doen met een inconsistentie?
Je probeert hier de oorzaak te ontdekken, en je probeert te classificeren, en volgens mij kan je ook niet zo heel veel meer doen. Dat biedt voldoende houvast, in deze stap in ieder geval.

· Helpt de stap ‘handling’ je met het afhandelen van een inconistentie?
Als je diagnose goed hebt gedaan, dan is dit in feite een invuloefening. het is geen rocket science, dus het is prima zo. Dit is een redelijk makkelijke stap.

· Bieden de contextfactoren en de profielen goede houvast met bepalen hoe je de methode uitvoert? Zie je het toegevoegde nut? *Het is heel belangrijk om vast te stellen waarmee je eigenlijk nou bezig bent. in feite zijn dat je prerequisites, de zaken waarmee je te maken krijgt, de zaken die je op voorhand moet vast stellen. meestal is het gewoon een gegeven. het is wel goed om dat in je methode vast te leggen, want dan weet je gewoon hoe je de rest van de methode kan werken.*

· Zou je de methode de volgende keer gebruiken?
Proberen zeker wel, in gebruik nemen .. ik ben altijd afhankelijk van anderen als ik hem in gebruik wil gaan nemen. Als ik bij klanten zit ben ik ook afhankelijk van klanten waar ik op dat moment zit. Daar moet die methode dan gebruikt kunnen worden. JE hebt natuurlijk altijd te maken met een omgeving waar je terechtkomt waar je niet degene bent die bepaalt wat er gebruikt wordt veel is er vaak al awat, en dan kun je wel langzamerhand een methode invoeren. Maar hij moet ook wel in een bedrijf ingebed worden en dat kost tijd, en tijd is geld, en dat zal dus niet altijd makkelijk zijn. Maar als ik architect spreek, dan zou ik zeggen: deze methode biedt mij wel extra houvast om mijn werk te kunnen doen, in die zin zou ik die methode wel willen gebruiken. Ik zou er in ieder geval ervaring mee willen opdoen.

· Voor welke andere toepassingen/domeinen zou je hier nut in zien?
Die twee doelen .. -> begin methode

Strength and waeaknesses:
Als ik hiermee zou moeten werken zou ik denk ik wel tevreden zijn en misschien komen er in de loop van de tijd wel dingen langs waarvan ik zeg dat zou beter moeten. Maar geen enkele methode die op de markt komt is 100 procent, en het duurt altijd even voordat je je methode zo goed als af hebt. Dat kost gewoon tijd.

Maar het geldt wel, dat dingen voor mij niet kunnen kloppen, terwijl ze voor een ander wel kloppen. Dat is echt context afhankelijk.

Strength:
Een groot voordeel van de methode is dat je hem gedurende alle fases van je project kunt uitvoeren, en afhankelijk van je werk kunt toepassen. En afhankelijk van de fase waarin je zit zul je meer nadruk leggen op 1 stuk of op een ander stuk. In die zin vind ik dat wel een sterk punt. In die zin is het toepasbaar.

Obstacle:
Wat ik net al zei, het uitrollen binnen een bedrijf, dat is het lastige punt want uiteindelijk is het een methode - er zijn veel stakeholders, een architect, een gebruiker, een developer. Uiteindelijk moeten zij met het systeem om kunnen gaan, meedenken met het proces, en ook kunnen begrijpen van hoe staan we er nou voor. Dus dat is heel belangrijk, tool adoption en tool understanding. Dus zelfs al gebruik je een spreadsheet, hoe krijg je die te gebruiken.

Ik zou het goed vinden als ze overzicht krijgen van hoe staan we er nu voor. Alleen dat moet dan wel op een manier zijn dat het in hun beleving wat zegt. En ik weet nou niet goed hoe je dat inzichtelijk moet maken voor gebruikers. Uit een concern kunnen ook een aantal vragen of mededelingen voortkomen. Misschien is dat iets om naar te kijken, daar kan een gebruiker wel wat mee.

Je moet je goed bewust zijn van de rol van de stakeholder, binnen jouw systeem, en dat is denk ik het belangrijkste. ITers snappen dat allemaal wel, maar de niet-technische stakeholders moeten het ook kunnen snappen.

als je zo'n conflict of inconsistentie hebt ontdekt, dat je dan zegt van de betrokken stakeholders, dit kan niet opgelost worden als die stakeholder niet is betrokken. Dus dan moet je hem betrekken. *dit kan je natuurlijk ook doen met zo'n freetext veldje.*

Heb je moeilijkheden ervaren met cognitie, hoeeveelheid informatie?
Het is natuurlijk lastig te zeggen, hangt van elke architect af, maar ook van zoiets stoms als de schermgrootte, hoeveel concerns je kiest, etcetera. Maar je moet sowieso niet teveel concerns kiezen, anders dan schiet het z'n doel voorbij: "het heet ook shortlist he, en dat heet het niet voor niks!"

Expert 4, EXPERT INTERVIEW, 04-03-2016, 16:00 uur

Welkom. Uitleg methode:
Vragen walco in schuingedrukt aangegeven

Uitleg methode, 2 disciplines, 3 fases, 7 activiteiten met substappen. *Is dit een proces wat 1 persoon doormaakt, of meer eigenlijk een organisatie?* Dit is een methode die een architect initieert als het ware, met eventueel collega's die hij erbij betrekt, en later stakeholders. *Inconsistentie is iets wat ook makkelijk ontstaat als je dus meerdere applicaties integreerd, waardoor er verschillende architecten op heb assigned, en dat is dus een major punt waarop inconsistenties ontstaan. In principe is dit dus een proces wat je als organisatie zou moeten doen - ik snap wel dat dat nu even niet goed te modelleren is en dat …* - Dus zit zou je boven de architecten moeten initieren? En dan met verschillende architecten moeten doorlopen zeg je? De focus ligt nu in ieder geval op het intiieren van de methode door 1 a 2 architecten. De eerste activiteit, planning, begint met het stellen van een doel. <uitleg doel> <uitleg scope> <uitleg stakeholders> *Zijn dat dan interne, externe, 'slachtoffers' van je software, of zijn dat business impacts?* Dat hangt heel erg af van de scope die je gekozen hebt en de reden met welke je de methode uitvoert.. *Gewoon eigenlijk de mensen die last hebben van de inconsistenties eigenlijk?* Ja zo kun je het wel zien. Je moet wel duidelijk in het achterhoofd houden dat er ook gebruikers bij de evalutie betrokken moeten kunnen worden, architecten of developers. Dan ga je naar de volgende stap <prioritering> <specificeren> *Je zou ze eigenlijk moeten opschrijven zoals je user stories opschrijft.* <matrix bouwen> <overlaps> *Participant vindt het niet duidelijk wanneer iets nou een conflict is en wanneer er nou een overlap is en wanneer er nou een inconsistentie is.* Matrix: op zich is dat nog wel een aardig punt - we hebben een aantal vaste perspectieven. <classificatie> <diagnose> *Participant vraagt wat voor soort inconsistenties er zijn nav 'type inconsistentie'.* Model inconsistenties, tradeoff inconsistenties. <handling> *We hebben een soortgelijk systeem voor bugs. Ik zou ze trouwens andersom plaatsen bij solving strategies. Postpone is eigenlijk een Ignore waar je nog geen afscheid van kan nemen.*

35:00 min

1. *De stappen en richtlijnen zijn wel duidelijk. Je hebt dat natuurlijk eromheen verteld.*
2. *- geen commentaar*
3. *ja, dat is zeker waar.*
4. *De gebruiker moet centraal staan, zeker in scrum. We gebruiken dus altijd een stakeholder of een slachtoffer in het geheel. Dus het lijkt me essentieel om die erbij te betrekken, eigenlijk zeg je dus van met de uitkomsten van dit proces, - we inviten eerst stakeholders, en later zeg je die halen we er niet bij.*
5. *Een design rationale en een kennisbank zijn effectieve manieren om kennis te hergebruiken tijdens deze methode.. Tja dat weet je natuurlijk nog niet. Ik zou er voorzichtig een vier op geven.*
6. *De methode past binnen mijn manier van werken - het past altijd, want hij is te tailoren. Ik zou er een vier op geven.*
7. *De methode voelt beperkend en dwingend aan, nee helemaal niet zelfs.*
8. *De methode beperkt mijn vrijheid als architect niet, nee ook niet, 5. Het is namelijk een stuk gereedschap wat je al dan niet in je hand neemt.*
9. *nee*
10. *prima*
11. *Even kijken, dat weet ik nu nog niet echt goed. Even kijken, betrekkende op een voorbeeld, die user registratie story die ik in mijn hoofd had om hierbij dan te gebruiken. Daar heb je natuurlijk - usability dus hoe makkelijk is het voor een gebruiker om zichzelf te registreren versus kan een hacker geen details ontdekken over huidige gebruikers en proberen in te breken, dus security als perspectief. Dan hebben we usability en security. Die leveren dus zeker concerns op en*

> *inconsistencies. Je hebt bijvoorbeeld een requirement dat een gebruiker makkelijk moet kunnen inloggen. Aan de andere kant moet een attacker niet achter een bestaande username van ons systeem kunnen komen. Aan de ene kant usability. User kan eigen gebruikersnaam kiezen is een requirement dat van belang is. Aan de andere kant moet een hacker geen bestaande usernames kunnen achterhalen. Gebruiker moet makkelijk in kunnen loggen. Een hacker mag geen persoonlijke informatie aan de weet kunnen komen.* Dat is dus meteen een vraag waarmee ik op dit moment stoei, concerns zijn voor elke architect net iets anders, dat is het lastige. Dat komt omdat ik de architect daar vrij in wil laten. Het gaat erom dat hij de bepaalde bouwstenen waar hij zich zorgen om maakt de bepaalde eisen - het zijn eigenlijk high level requirements maar die voor de architect juist superbelangrijk zijn voor de architectuur en de gedachtegang is dat hij de voor hem belangrijke concerns als bouwstenen neemt om daar de inconsistenties in te ontdekken. Omdat de plaatsen waar concerns overlappen, die inconsistencies ook vaak de meest schadelijke gevolgen hebben, want anders heb je ook geen concern. De vraag is, dit zijn requirements, misschien moet ik hier wel gewoon een perspectief neerzetten met hieronder requirements. Mijn concern is dat medische informatie naar buiten lekt doordat een hacker toegang krijgt tot het systeem en ongeauthoriseerde informatie wordt vrijgegeven. En de ander zou kunnen zijn is dat een gebruiker het systeem niet gaat gebruiken, gebrek aan adoptie. En als je ze tegen elkaar af zou zetten? Op zich is dat wel een betere, als je het zo formuleert dan komen die business value en die classificatie aspecten erbij. Er is natuurlijk een enorm risico voor de business als onze medische database gehackt zou worden, maar de kans is niet zo groot. Maar de adoptie van het systeem is natuurlijk ook, als dat niet goed gaat dan - heb je ook een enorm business risico. Alleen is de waarschijnlijkheid daarop kleiner of daar heb je in eider geval invloed op. Dat is wel een goede dat je dat op deze manier brengt.

Als je de architectuur op een bepaalde manier door een bepaald perspectief bekijkt, dan heb je natuurlijk een bepaald aantal zorgen en die vertalen zich eigenlijk ook weer in de requirements die je opneemt. en de requirements die vormen op zich zelf ontwerpbeslissingen. Eigenlijk is een inconsistentie tussen ontwerpbeslissingen uiteindelijk hetgene waar je last van kan krijgen.

Je kunt dit misschien ook wel doen op basis van conflicterende requirements. Of moet ik in de methode zeggen: de concerns zijn de requirements die voor de architect belangrijk zijn? *In weze is een concern een van de bouwstenen waar de architect dan mee werkt.* Misschien moet ik dat als defintie geven. Concerns zullen voor jou een andere invulling hebben dan voor een architect bij een ander bedrijf. *Ik denk dat we hier in het midden uitkomen, ja wellicht een vier…*

Een matrix is een goede en overzichtelijke manier om dat tegen elkaar af te zetten, performance, 4-5 .. beetje het zelfde.

Vond je de situationele factoren nuttig/waardevol?

Wat het wel toevoegt is - van nature zou ik gewoon meteen naar die pragmatic hoek gaan, maar als je kijkt naar wat voor type organisatie we izjn dan sluit dat er wel op aan, ik denk dat de linker hoek wel een beetje aan het verwdijen nis maar daar heb je zelf beter zicht op denk ik . Alleen de implementatie van het pragmatische model dat heb je aangegeven dan kun je het gewoon in een sessie doen, en met alle stakeholders direct afkaarten en dan heb je die tool niet zo nodig, maar wat de systematic approach heb je dus waarschijnlijk wel jouw hele workflow apparaat nodig, en bij balanced, kun je dus zelf kiezen, dan pak je dus die spreadsheet erbij als tussenweg, of kies je de stappen die voor jou systematischer moeten en of pragmatischer. Dan pak je eigenlijk op je eigen manier de profieltjes die je per stap nodig hebt.

Zou je dat nuttig vinden? Zou je dat helpen als architect? *Ik zou het zelf niet gebruiken, ik zou meer gewoon direct aanvliegen en beginnen. Meeste architecten weten denk ik van zichzelf wel dat ze in een bepaald profiel passen.*

De diagnosestap, heb je die ervaren als nuttig? De cause vaststellen wel, maar de type, stelde ik nog een vraag over. De classificatie van de inconsistency is met nog niet helemaal helder (participant doelt op type inconsistencie). Wat je in deze stap doet? *Wat voor type kan ik aan een inconsistency hangen dan?* De stap in de classificatie is dat je dus kijkt wat voor business value, wat voor impact, en wat voor engineering effort er bij komt kijken. **Ooh, die stap is duidelijk, maar die classificatie van het type inconsistentie nog niet.*

Eigenlijk moet je beginnen met is het een incident, komt het vaker vvoor? Als het een incident is, hoef ik het niet vaker meer te doen. Oke, dat is wel een goede. Moet die bij impact staan? *Nou ja, dan is eigenlijk gewoon de risk laag. Ja trouwens, het gaat om het risico op de volgende inconsistentie op die manier he? Dat is wel een apart dingetje natuurlijk, die moet in zn eentje bovenaan. Als je iets hebt wat niet verder voorkomt dan hoef je misschien niet veel rootcause analysis te doen, maar de vraag is of je een antwoord kan geven wanneer je niet weet waar het vandaan komt en wat voor ding het was. Is nog een beetje fuzzy vind ik, zou je echt een paar keer zo'n ding moeten doorlopen.*

Dit ken ik - dit is iets wat ik van uit - of wat ik eigenlijk impliciet doe bij het behandelen en prioriteren van bugs, dus dit is eigenlijk hetzelfde proces, alleen is dit een ander abstractieniveau.

Zou je de intentie hebben om deze methode te gebruiken/uit te voeren? *Ja het lijkt me op zich nuttig om - ik zou er wel een heel simplistische versie van inzetten, maar ehm, we hebben zelfsturende teams die ook hun eigen designs doen, en we hebben een soort van definition of to-do waar een aantal van deze - of eigenlijk dezelfde - waar we beogen mee dezelfde doelen te halen als die je hier ook hebt, dus hoe ik zeg maar vanaf verschillende perspectieven naar de 'story' te kijken en te zien hoe vertaalt zich dat dan in requirements of hoe los je die conflicterende requirements op.*

Ik denk dat het op zich niet moeilijk is om dan een aantal prefab perspectieven op te schrijven en daar dan dus eigenlijk de requirements in dat perspectief op te schrijven en dan misschien inderdaad zo'n matrixje is natuurlijk niet zo heel moeilijk in te voegen. Dus, op storyniveau is het denk ik wel goed inzetbaar. Als overall-methode om productgrenzen heen inconsistenties te vinden, is wel veel lastiger, en dan kom je eigenlijk ook tot de kernvraag dat je als heel product/bedrijf moet je dat dan met een board of een groep architecten doen en dan kom je meer op jouw level tooling uit waarbij je iets formeler kan opschrijven maar dat moet dan zeker niet op - dat moet dan echt op heel abstract niveau want anders verzuip je in de hoeveelheden concerns.

Daar maak je die shortlist dus ook voor, ik kan me heel goed voorstellen dat als je de scope veel breder trekt dat het algauw een stuk lastiger wordt.

Op storyniveau zie ik het al gauw gebeuren en ook voordelen hebben op wat we nu doen, dat we eigenlijk de teams zelf schrijft/werkt de story verder uit tot in detail. maar bijvoorbeelde de hazard analysis of zo die bungelt er nu een beetje achteraan, maar als je nu een team een tool in handen geeft zo kun je zelf een initiele analyse doen, zonder dat het heel ingewikkeld is, dan kun je voorkomen dat het uitstapje naar dat hazard analyse board gedaan hoeft te worden.

58:00 Ik denk dat het voor de architectuur zelf niet zo … Ik zou het zo 1 2 3 niet echt kunnen zeggen, ik denk dat deze vraag beter te beantwoorden is als je de methode hebt doorgelopen.

Strenghts
Wat zijn volgens jou de verwachte voordelen van de CCIM methode?
Ik denk dat als je het goed doet dat je gewoon minder zwabbert tijdens het implementeren van features. Want nu ga ik - het punt is dat dat ontdekken van concerns dat gebeurt eigenlijk overal en ook als je bezig bent met het implementeren kom je er soms achter dingen, maar als je die dingen nou eerder vindt en addresseert, dan scheelt dat bijvoorbeeld rework bijvoorbeeld. Dat zie ik wel als groot voordeel.

En misschien inderdaad ook wel meer zelfstandigheid voor de development of het team zelf om hun eigen problemen tegen elkaar af te wegen. Dus dan zou je eigenijk zeggen dat een andere toepassing van de methode is dat je eigenlijk de tool aan de teams geeft met daar met betrekking op hun scope op hun stukje product/hun architectuur. En dat ze zelf de belangrijkste concerns die voor hun belangrijk zijn, zodat je iedereen ongeveer binnen het gareel houdt en zelf al heel veel afwegingen doet.

Wat zie jij als zwaktes aan de methode? Obstakels? *Dat zijn weer andere dingen. het is een beetje zoals prince2 ofzo, dat je als je de tailoring niet doet, dan is het te vaag om in te zetten. Dus je moet tailoring doen, dus je moet eigenlijk zeggen, voor onze organisatie gaat het zo werken en dit is een concern. En dit niet. Daar hebben in het begin met hazard categorieen ook heel sterk gehad, als je de type concerns niet goed - of als je niet goed definieert wat een concern is dan blijf je heel veel discussies hebben of iets nou wel of niet een concern is. vandaar dat ik ook wat meer naar conflicting requirements trek, omdat requirement wel een begrip is waar iedereen mee kan werken. En wat dus van concerns wel weer een voordeel is, is dat je er bij die assessment hier wel een gewicht aan kan hangen, en kan zeggen van de individuele gebruiker is wel belangrijk, maar de security van het systeem is nog belangrijkere factor. Daar kun je makkelijker gewicht aan hangen. Dus de definities goed krijgen, zou een eigenlijk een meta-activteit voordat je begint moeten zijn. Omdat die tailoring zo belangrijk is, zeg jij, moet je dat ook degene die hem doet laten doen.*

Ik zat zelf nog te denken aan het geven van een voorbeeldcyclus, of een voorbeeld template, dus eentje voor de pragmatische, eentje voor de systematische en 1 voor de balanced.
Daar kwam ook al uit, als je iets zou veranderen, wat zou je dan veranderen?

Hoe zou je de user stories gebruiken om met concerns samen te werken? Ik kan je wel wat laten zien:

Definition of done:
Defintion of to do:
Definition of ready:

Eigenlijk zou ik in 1 van deze stappen - oh dit is ook de definition of ready. Dit is de definition of todo: een checklist wat je moet doen en waar je op moet gaan letten als je een nieuwe story begint.
Hier heb je ook de verschillende, backward compatibility, databases, performance, sizing statistieken, auditing, system, performance, dus dat zijn een beetje die verschillende perspeciteven. ik zou ze niet allemaal los willen zien, maar als je heel veel gaat auditen, dat heeft natuurlijk veel invloed op performance. Security heeft natuurlijk weer impact op juist - ook op performance. Op weer andere aspecten. Licensing heeft ook weer invloed op performance, of juist op usability. Usability zit hier trouwens nog niet echt in. Dit is nu het puntje, wat nu dus relatief ‘oh denk hier over na’ is, maar denk ik wel de moeite waard is om dat af te wegen tegen elkaar. Zodat het team zelf kan zeggen, ik heb dus hier een conflict ofzo, en als ze er niet uitkomen met het toekennen van het gewicht dat ze aan het architect board vragen, we hebben hier dit vastgesteld dat er conflicterende requirements waren, wat is nou het belangrijkste, hoe kunnen we er omheen werken, of hoe kunnen we voorkomen dat het echt als conflict de software in gaat?

Definition of ready, to do? Waar staan die voor? Todo is vooraf een user story, om de taken op een rij te krijgen die je moet doen om de story goed te implementeren, dat is ook in het proces dat je de requirements in de story goed krijgt, de user story zelf goed krijgt, dus de tagline, en voor wie je het maakt en waarom.

Defintition of ready is op het moment dat de tester ermee aan de slag kan,dus op het moment dat je echt duideijk hebt wat je gaat bouwen. Dan gaat de de tester zijn testcase schrijven, en zijn testscenarios of testgevallen vaststellen. Op het moment dat de story read is kan hij daar echt me aan de slag. Dat is ook een kleine review geweest met de product owner. En dan gaat ie verder. Bij het afsluiten van de story, dus als hij verified is, dan moet hij nog geaccepteerd worden en dan worden zeg maar die formele dingen langsgelopen, en dan loop je alle dingen van definition of done na. Is die security analyse goed gedaan? Is die concern analyse goed gedaan? is alles wat aan techdebt of … - definition of done heeft wat meer validation dingen ook, is er een demo loads gedaan, heeft een klant iets gezien en gezegd dat hij het goed vond. Is zeg maar alle administratie afgewerkt? Open issues zijn dus de bugs die eruit komen, vinden we het belangrijk genoeg om te fixen, nee, ja, kan het worden gepostponed. CCB staat voor change control board.

Hazard analysis. Daar hebben we een speciale database voor, wat er eigenlijk gebeurt nu is dat er een team bij het - dus we hebben een aantal categorieen van hazards, dus dit komt dit in de buurt van hoe je concerns gezien hebt in eerste instantie. het begint met een hazard brainstorm, meestal gewoon een paar ervaren mensen die weten wat ze aan het doen zijn. dan is er gewoon een klein comite wat kijkt is het belangrijk genoeg om iets mee te doen, of niet. Dan wordt er een hazard van gemaakt. De hazard wordt dus gelinkt met iets in de codebase, zodat je aan kan tonen we hebben iets met die hazard gedaan. *Zou dat dus je design rationale kunnen zijn?* Dus bij ons is bv patient privacy een heel belangrijk concern en daar hebben we hier dus een aantal verschillende hazards die daar mee te maken hebben. En onder bijvoorbeeld ‘gain information about geexporteerde beelden naar een ander systeem zonder een expliciete reden’, je ziet ook wie dat gereport heeft. Er hangt dus een source feature (bron naar een feature in de code die die hazard mitigeert) en omdat we uiteindelijk een bepaalde asseessment hebben gedaan, van nou daar komt niet zo heel waarschijnlijk voor, is er geen echte mitigation aangezet volgens mij. Maar op het moment dat hij boven een bepaalde treshold komt dan moet je er wel iets mee doen. Even kijken dat kan ik ook nog wel laten zien.

Ik kan natuurlijk absoluut niet inloggen, maar hoe hebben jullie nou zo’n hazard geclassificeerd? En een subhazard? En waar kan ik dus van leren? En hoe evalueer / classificeer je die? Ik zag dat jullie probability en severity hadden. *Dus de kans dat iets optreed, - kijk als iets als je hebt wat catastrophale gevolgen heeft dus de patient zn been wordt er afgehaald of hij komt te overlijden, dat is natuurlijk catastrophaal. Maar de kans dat dat gebeurt door gebruik van onze software, is natuurlijk heel erg klein. Als dat vermenigvuldigt, en dat blijft onder een bepaalde treshold, dan doen we er eigenlijk niks mee. Maar als er iets is wat regelmatig kan gebeuren en toch wel vervelende gevolgen heeft dan komt het boven die treshold uit en dan doen we er dus iets mee. De probability keer de severity is de assessment value.*

Dit is ons hazard register, dit wordt geexporteerd vanaf die database die je net zag, dit gaat over de huidige release, dit gaat over welk product betreft, van wat de initiele exposure is. Je hebt iets wat probable is (wat zeker kan gebeuren) en waarvan de severity ook nog eens significant is, dus dan moet je iets mee doen met een design constraint. Dus hier staat dan je hebt een manier om patient demographics te zetten door dingen op de URL toe te voegen. Daar kun je natuurlijk gewoon een mismatch hebben tussen de data die gedisplayd wordt en de patient. Dus dat is inderdaad een dingetje dat gedaan moet worden, en dan ziej e dus dat er in een feature mitigation is gemaakt voor dat probleem. Dus soms heb je ook dingen die in de configuratie bijvoorbeeld een issue is, dus je hebt hier de configuratie eigenlijk fout gezet, dan raak je eigenlijk informatie kwijt, dat is een outcome die ik ook

niet wil, dat kan occasiional zijn, en de uitkomst is significant. En daar moet je dus ook iets mee doen, en hier is dan gekozen om er een label van te maken. (ameliorate) in plaats van een protective measure in de software wat een zwaarder en duurder - meer engineering effort is is er gekozen om in de manual ervoor te waarschuwen, een soort van ameliorate.

Het is heel erg gericht op een subcategorie van concerns, dus performance is eigenlijk … het is eigenlijk de categorie system unavailable maar het is niet zo dat het trager is dan drie seconden, dat niet. Dus het niet zo specifiek als een requirement. Met concerns zeg je eigenlijk van hij mag er niet lang uit liggen, en met requirements heel specifiek mag zijn. Die categorieen hier zijn dus wat iets grover: als het systeem slow is hoeft er nog niet te worden ingegrepen, maar system unavailable is natuurlijk gewoon iets wat we altijd willen voorkomen. Zou het kunnen zijn dat ik hier een voorbeeldje van kan krijgen?

H.4 Protocols

DEVELOPMENT: INTERVIEW PROTOCOL

Definities
Concerns: A concern about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture. Concerns can come from every stakeholder, and from any domain.

Inconsistency: een situatie waarin twee beschrijvingen niet onderling compatibel zijn en niet aan de relatie voldoen waarvan is vastgesteld dat deze er zou moeten zijn. Dit kunnen diagrammen zijn, modellen, producten in fases van een proces, of assumpties en beslissingen. Voorbeelden: als in 1 use case diagram het woord klant wordt gebruikt, en in een class diagram het wordt 'lener'.

Momenteel doen we onderzoek naar inconsistentiemanagement in de softwarearchitectuur, en hoe zgn. concerns <uitleggen concerns> kunnen bijdragen aan het lokaliseren van inconsistenties.

Wat gaat er gebeuren?
De informatie uit de interviews wordt samen met een literatuurstudie gebruikt om een methode te ontwikkelen die het voor de SA mogelijk maakt om inconsistenties op een gestructureerde manier te lokaliseren. We gaan er vanuit dat de architectuur al in kaart is gebracht, of dat in ieder geval de concerns/requirements al redelijk zijn geidentificeerd. Het ontwerpen van een architectuur valt dus niet binnen de methode. Hypothese: door op basis van zgn concerns naar inconsistenties te zoeken, hopen we de SA hulp te bieden door de pijnpunten te laten zien waaar inconsistenties mogelijk kritisch kunnen zijn.

Begeleidende optionele tekst:
Het consistent houden van de architectuur kan op meerdere manieren, bijvorbeeld: het identificeren en aanmaken van enorme regelsets, om zodoende elk element via regels aan een ander element te linken. Als er een regel wordt gebroken dan indiceert dat inconsisentie.

De zwaktes aan deze methode zijn 1) dat alles gedocumenteerd moet worden, 2) dat er voor elke combinatie van elementen een regel aangemaakt moet worden, en 3) dat er vaak gebruik gemaakt moet worden van formele wiskunde, die lastig is, en slecht schaalbaar is.

Dit werkt in de praktijk niet goed, en wat nodig is, is een gestructureerde aanpak om inconsistenties te vinden in de architectuur. **Vraag:** *Hoe kan een software architect op een gestructureerde manier geholpen worden om inconsistencies te vinden in een software architectuur?*

Dit is lastig, aangezien elke architectuur totaal anders is en compleet afhankelijk van de eisen (concerns/constraints) die er gesteld zijn aan zijn architectuur. Daarom een methode die de architect kan gebruiken om op een gestuctureerde manier zijn eigen architectuur te doorzoeken op inconsistenties.

Oplossing:
Een methode die het de architect mogelijk maakt om op bepaalde 'pijnpunten' naar zijn architectuur te kijken. Op deze manier kan hij gestructureerd de architectuur nalopen (documentatie of fysieke systeem) en op plekken kijken waar inconsistenties het meest kritiek zijn.
Die pijnpunten worden gevonden door te kijken waar bepaalde belangen (concerns) met elkaar overlappen. Als belangen met elkaar overlappen, hebben ze een relatie, en op de plekken waar belangen gerelateerd zijn zouden dus mogelijke inconsistenties kunnen optreden.

Toegevoegde waarde:

Een architect wordt niet gedwongen alles te documenteren, hoeft geen gebruik te maken van een formele methoden, kan alleen checken welke belangen hij denkt dat belangrijk zijn. Het past dus in de agilewerkwijze. Het doorlopen van de methode levert een framework op, met daarin de belangrijkste belangen vanuit bepaalde perspectieven. Hij kan dit framework aan het begin van de levenscyclus maken, en het dan gebruiken als check. Het eindproduct kan gebruikt worden door het gehele team om te kijken waar belangen elkaar 'raken'.

Hoe werkt het:
Het doorlopen van de stappen van de methode laat de software architect en zijn team nadenken over bepaalde belangen die ze van te voren hebben geidentificeerd en hoe deze zijn verwerkt in de architectuur. Hij selecteert bepaalde belangen die belangrijk zijn op dat moment, en mapt ze uit tegen andere belangen. Per cel van het framework gaat hij de belangen af, kijkt: 1) hoe ze gedocumenteerd zijn, 2) of ze gedocumenteerd moeten worden, 3) hoe ze de 2 belangen zich verhouden, 4) welke ontwerpbeslissingen hij heeft gemaakt, en 5) of er inconsistenties zijn met betrekking tot 2 specificaties van die belangen.

Vragen
Consistentiemanagement is ruwweg op te delen in de volgende fases: voorbereidingen, monitoren van inconsistenties, diagnose van inconsistenties, het behandelen van inconsistenties, en het monitoren van de gevolgen van een actie.

Het eerste gedeelte van dit interview gaat over inconsistentiemanagement in de praktijk.

1. Kun je kort je rol als SA beschrijven?
2. Op welke manier worden inconsistenties in de architectuur (dus zowel in de AD als in de architectuur zelf, bv ontwerpbeslissingen) gemonitord en gecheckt?
3. Op welke manier classificeren jullie gevonden inconsistenties?
4. En op welke manier handelen jullie deze af?
5. Over welke van de hiervoorgenoemde stappen ben je momenteel tevreden en vind je dat ze optimaal zijn?
6. Zijn er momenteel dingen waar je ontevreden over bent als het gaat om (in)consistentiemanagement?

Graag zou ik nog willen weten:

7. Koppelen jullie <concerns> aan ontwerpbeslissingen in een soort van architecture rationale?
8. Als jullie dat niet doen: denk je dat dat van toegevoegde waarde is? En hoe zou je dit doen?

Het idee is als volgt: <<lees stuk voor>>

Dit omdat het gebruiken van tools, en formele methoden in de praktijk vraagt om een hele hoop documentatie, bv diagrammen en modellen.

9. Als we concerns als first class objects gebruiken om een framework te creeeren waarin de architect en zijn team kunnen kijken om mogelijk inconsistenties te ontdekken, hoe zou je dit dan doen?
10. Stel je dus een cel voor, <kruising van een kolom en een rij>, welke stappen zou je zetten om te kijken of hier inconsistenties te lokaliseren zijn?

Dus bv: per cel: de diagrammen of documenten waarin de concerns zijn opgenomen checken.

Inconsistenties hoeven niet per se heel erg te zijn, maar op de plekken waar concerns elkaar overlappen, kunnen inconsitenties wel vaak vervelende gevolgen hebben. Vandaar de keuze om concerns te gebruiken als leidraad (first class citizens) in deze methode.

11. Als we concerns gebruiken als input voor de methode:
    a. hoe zou je deze prioritizeren?
    b. Hoe zouden we deze kunnen 'verpakken'?

12. Wat is er voor nodig om de methode goed in je dagelijkse activiteiten te laten passen, en vooral, niet te veel overhead veroorzaken?
13. Op welke andere manier kunnen we deze methode gebruiken?

Denkt u bv aan het gebruiken van het framework als checklist als de architectuur gebouwd wordt? Of als communicatiemiddel/tool binnen (en tussen) de teams?

EVALUATION: PROTOCOL

Definities
Concerns: A concern about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture. Concerns can come from every stakeholder, and from any domain.

Inconsistency: een situatie waarin twee beschrijvingen niet onderling compatibel zijn en niet aan de relatie voldoen waarvan is vastgesteld dat deze er zou moeten zijn. Dit kunnen diagrammen zijn, modellen, producten in fases van een proces, of assumpties en beslissingen. Voorbeelden: als in 1 use case diagram het woord klant wordt gebruikt, en in een class diagram het wordt 'lener'.

Momenteel doen we onderzoek naar inconsistentiemanagement in de softwarearchitectuur, en hoe zgn. concerns <uitleggen concerns> kunnen bijdragen aan het lokaliseren van inconsistenties.

Wat gaat er gebeuren?
De informatie uit de interviews is samen met een literatuurstudie gebruikt om een methode te ontwikkelen die het voor de SA mogelijk maakt om inconsistenties op een gestructureerde manier te managen. We gaan er vanuit dat de architectuur al in kaart is gebracht, of dat in ieder geval de concerns/requirements al redelijk zijn geidentificeerd.

<Uitleg methode uit Protocol interviews>

CASE STUDY
Om een goed beeld te krijgen van hoe de methode in de praktijk gebruikt kan worden, willen we 2 lichtgewicht case studies doen, waarbij een gedeelte van de methode uitvoeren.

Voorbereidingen:

Architect:
- finding out which part of architecture yields interesting results,
- choose a selection of concerns with which you are currently struggling.

Jasper:
- document study, architecture study, concern / requirements documents

Uitvoering:
- Time: 1,5 to 2 hours
- Explain every step and deliverable as we go through the phases of the CCIM method.
- Illustrate the simple version of the method along with the high level deliverables.
- Have a moment in which participants can ask questions and ask clarification.
- Execution of an instance of the method.
  - However, with a limited amount of concerns, virtual stakeholders, and a small piece of the architecture of the company.
- Observe whether they experience difficulties during execution; pay attention to the method requirements.
- Ask the questions + statements

INTERVIEW
- Time: Interview of 45 minutes / 1 hour.
- Walkthrough of several screenshots of deliverable/tool
- Explain every step and deliverable as we go through the phases
- Illustrate the simple version of the method along with the high level deliverables

- Have a moment in which participants can ask questions and ask clarification.
- Ask the questions.

QUESTIONS:

(1 = Totally disagree .. 5 = Totally agree) (- is negatively phrased, + is positively phrased)
Statements (Requirements satisfaction) + Comments:

1. + The guidelines and process support is clear. (AE1)
2. + It is clear in which stage the method should be applied. (AE2)
3. + I have a technique for each step if it is required. (AE2)
4. - It is not feasible to identify and involve stakeholders in this process. (AE3)
5. + A design rationale and a knowledge base are effective means to promote reusability in this process. (AE6)
6. + The method fits in my daily practices as a software architect. (AL3)
7. - The method feels restrictive and compelling. (AL4, ER5)
8. + The method does not constrain my architectural freedom. (AL4, ER5)
9. - The method is difficult. (ER1)
10. + The method is flexible. (ER6)
11. + Concerns are an effective way to detect inconsistency. (usefulness), AL1
12. + Using a matrix is an effective way of detecting overlaps. (usefulness)

PERCEIVED USEFULNESS:

- Did the 'diagnosis step' help you with deciding what to do with an inconsistency?
- Did the 'handling alternatives' help you with solving an inconsistency?
- Did the situational factors and profiles help with deciding how to apply the method? (AE4, ER2)
- Do you have an intention to use this method next time?
- For which other purposes/contexts would this method be useful?

STRENGTHS WEAKNESSES

- Are you satisfied with the (quality of the) outcomes? (ER4)
  - *Did you gain more insight in the architecture?*
  - *Do you have the feeling that you detected and solved more inconsistencies?*
- What are the perceived benefits of applying the method? (ER4)
- What are the perceived obstacles of applying the method?
- Did you experienced difficulties with the quantity of the information? (ER8)

IMPROVEMENTS:

- If you could change anything, what would you do differently?
- Do you have further recommendations?

NOT VERIFYABLE THROUGH QUESTIONS:
AE5: Validation: evaluation is performed.
AL1: Concerns: concerns are first class citizens.
AL2: Support views: views are required, so they are incorporated.
AL5: Tool support: have developed conceptual screens.
AL6: Incremental adoption:
ER3: Tool support
ER7: Incremental improvement and completion
ER8: Cognitive overlead
ER9: Hierarchy

ER10: Evolution
ER11: Accessibility