Incremental Method Enactment in Software Engineering Tools

MBI Master Thesis

IKU-3555135

Geurt van Tuijl

25 August 2012

Universiteit Utrecht

Incremental Method Enactment in Software Engineering Tools

A Method Engineering approach

Master thesis

Author:	G. J. (Geurt) van Tuijl
E-mail:	g.j.vantuijl@uu.nl
Student number:	3555135
Graduation period:	November 2011 – September 2012
Supervision:	prof. dr. Sjaak Brinkkemper and dr. Slinger Jansen
Thesis number:	IKU-3555135

In conjunction with

Universiteit Utrecht

Department of Information and Computing Sciences

Leuvenlaan 4

3584CE Utrecht

The Netherlands

FINAL VERSION

25 August 2012

Abstract

Today, organizations employ all kind of models or methods to support their software development. When a method becomes too complex or the need exist to enhance their method, a model is created and typically updated with new information. Often however, the model hardly complies with the actual working situation due to continuous changes in the actual process. This research presents a solution to maintain the integrity of the actual way of working in a software engineering tool consistent with the employed models. By developing a custom-made *enactment mechanism,* we incrementally update the interface of the CASE tool called Jira when an adaptation in the model occurs. The models are based on the notion of Process Deliverable Diagrams and we apply incremental method evolution to our models. Based on the evolution of the models, the lay-out of Jira is tailored to fit the new model. The results of this research show that it is possible to incrementally enact an increment in a CASE tool when a custom-made mechanism is in place. From this study, we can conclude that already employed CASE tools within organizations need to incorporate a great amount of flexibility and that Process Deliverable Diagrams need more fine-tuning in order to be suitable for process automation. In addition, the employed CASE tool needs to contain extensive workflow support. Based on our research findings and expert evaluations through interviews, we concluded that incremental method enactment was only desired in medium/large-scale organizations if the enactment mechanism provides a sufficient amount of functionality and adaptability and stays simple. Also, the enactment mechanism was perceived as very useful for companies that are already using Jira.

Keywords: incremental method enactment, method engineering, incremental method evolution, process deliverable diagrams, Computer Aided Software Engineering.

"We should help create the future, not just study the past"

Paul Gray in Kock *et al.* (2002)

Preface

As of 2003, I decided to study at a regional education centre where I chose to focus on Information Technology (IT). Even though I had by that time very little knowledge of what IT actually was and what it entailed, I easily obtained my diploma after attending classes for three years. The lectures were highly superficial and in order to obtain a more profound understanding of IT, I decided to continue my study at the University of Applied Sciences in Utrecht. I started with the study Informatics but after one year I decided to switch to Business Informatics as I noticed this was the field in which I wanted to enhance my existing knowledge. While I was trying to obtain my Bachelor's degree, I met three people with which I had the best time at the university. Later it turned out that we all agreed upon completion for a Master degree at the University of Utrecht. I knew it was going to be awesome again together, but I also kept in mind that a two year study is a rather long continuation after having already studied for 6,5 years. Still I managed to meet the requirements for granting access to the University and there it was, the first day at the University. I started at almost the lowest educational level and yet I managed it to make it to the University. Now, looking back on this period, I truly learned so much in all these years that I cannot even image what knowledge I had 9 years ago. Especially with regard to science and the fact that you are working with the state of the art knowledge is very satisfying and challenging.

Yet I did not manage to accomplish this by myself. First of all, I would like to thank Michail Theuns, Yasin Topcu and Thomas Geertshuis which all have provided me with the best time during my study. Their exceptional knowledge and great personality was a truly addition to all these years of hard labour. Secondly, I would like to thank my both parents for always being there and supporting me in making my own decisions. They always pushed me to strive for excellence and to exceed my own boundaries. Eventually I did. Finally, I really want to thank Sjaak Brinkkemper for all the valuable feedback, his' positive mind-set and the recommendations during this study. Also, I would like to thank Slinger Jansen for his' critical feedback on the draft version of this thesis. Finally I would like to thank my former teacher Inge van de Weerd for all her effort and dedication that she put into our paper, my other researches and the start-up of this Master thesis.

Geurt van Tuijl
Utrecht, 25 August 2012

Table of Contents

List of figures

List of tables

CHAPTER 1

Introduction

Today, organizations employ all kind of models or methods to support their business processes. Certain models are usually in place to represent how a certain process within an organization should be carried out. To suit the needs for their scenario better, the available models are usually continuously updated based on feedback and reflection from different stakeholders. Often however, the models scarcely comply with the actual working situation due to continuous changes in the model. Especially when a model within an organization becomes too complex or the need exist to enhance their existing model, the model is usually updated with new information or replaced with a new model. To improve their existing models, organization can employ best practices or they can develop a tailor made solution that is tuned to their specific situation. This latter solution is referred to as Situational Method Engineering.

Regrettably, a change in the model does not necessarily mean that users change their habits and way of working. Habits gained from previous models are often still in place causing a mismatch between the models and the actual way of working. As a result, vital information is lacking that is needed to support the development of their software. The updated model required additional data to be entered in a software engineering tool. However, this data may never been collected because users do not know this data is required or they do not even know there is an updated model.

1.1 Problem statement

The problem sketched above is subject of this research. The research in this thesis presents a solution to maintain the actual way of working in a Computer Aided Software Engineering (CASE) tool consistent with the employed process models. A CASE tool is usually in place to support the development of software by employing different techniques and tools. By means of an *enactment mechanism* we aim to update the interface of an employed CASE tool when a Computer Aided Method Engineering (CAME) tool adaptation occurs. A CAME tool is a computerized tool to support the process of creating (situational) methods.

By updating the interface of a CASE tool instantly according to the new process model, users merely notice the impact of a change in the process model within an organization. Users would notice that they are filling in other fields or providing the system with additional information. In fact however, the employed CASE tool is processing their data in another way. Moreover, the likelihood of a situation where users forget to enter the right data into the system is reduced because the associated software engineering tool conforms to the latest process model. Due to this conformance, employees are always providing the software engineering tool with the required data and hence organizations prevent a situation where vital data is lacking that may be needed in a later stage of the development of their software.

1.2 Context of this research

This research focuses on existing CAME and CASE tools. We propose a solution where the employed CASE tool conforms to the employed CAME tool. To achieve this, we need to link two architectural levels from the proposed architecture of Kottemann and Konsynski (1984) and ISO (1990). The architecture combines information use and evolution in four levels. In 2002, the Object Management Group created a first standard to this architecture and labelled the four levels as the Meta Object Facility (MOF) levels (Object Management Group, 2011). Figure 1 depicts the four MOF levels.

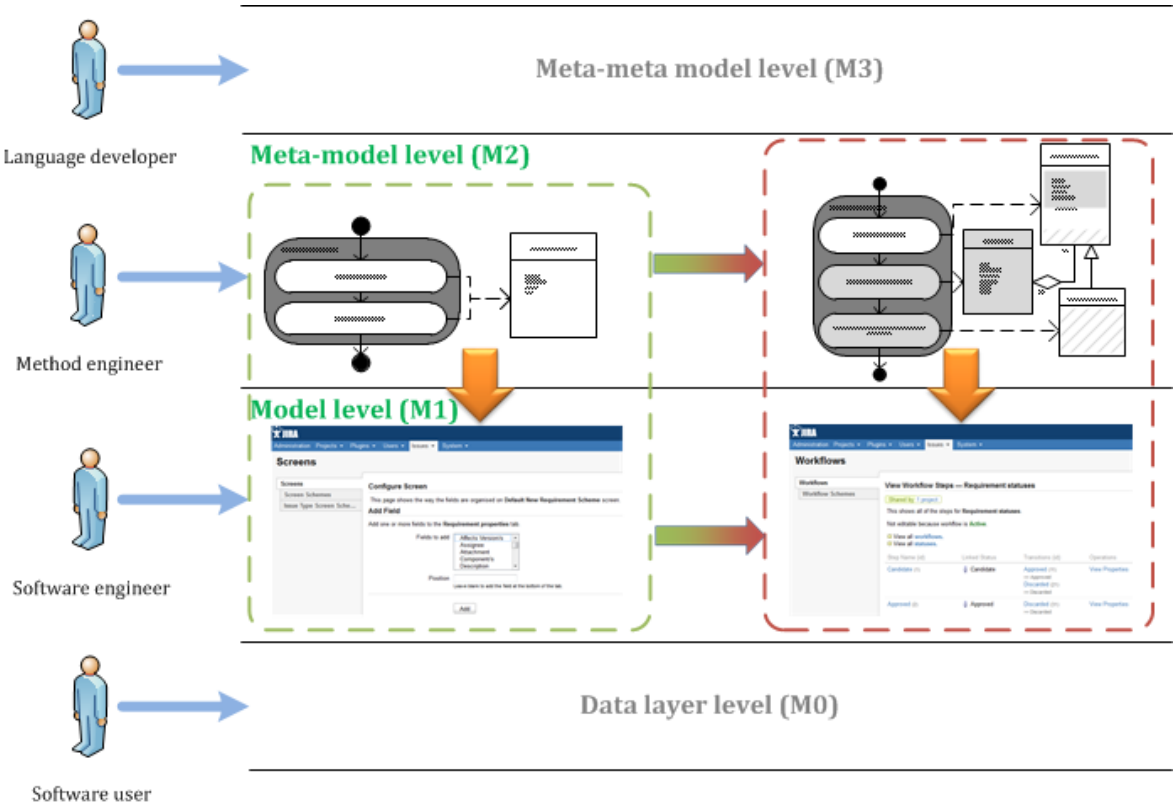


Figure 1. Four architectural levels for information use and evolution.

The top level is the *meta-meta model* level (sometimes referred to as the M3 level) and is a meta-model of a meta-model. The meta-meta model tells a method engineer what is allowed and what is not allowed with regard to its modelling principles. The meta-meta model level can hence be seen as a blueprint for the meta-model on a very abstract level. The lowest architectural level is the *data layer level*. The data layer level is the level a regular user that is using the software is concerned with. This can for instance be some fields on the screen in which the user needs to type personal credentials or other data is required in that particular field. The *meta-model level* is the level where meta-models are being created. A meta-model is an abstract model of a model that tells the method engineer what is allowed with regard to its modelling principles. For instance, you may connect an activity with an activity but not connect an activity with an attribute of a concept. During the development of a method, the method engineer should adhere to the meta-meta model. The *model level* is the level the software engineer develops its software.

The development is based on the meta-model that is created by the method engineer. It is a tangible elaboration of a meta-model and always complies with the *meta-model level.*

The focus of this research is placed on the *meta-model level* (the level in which CAME tools are employed and used) and the *model level* (the level in which CASE tools are employed and used); indicated by two orange arrows. The orange arrows represent the actual *enactment mechanism* that makes sure of whether the model level is in accordance with the meta-model level. In other words, the interface of the CASE tool corresponds with the model on the meta-model level. When the model at the *meta-model level* changes, the interface of the employed CASE tool at the *model level* automatically changes according to this new model. A change in the model is represented by the green/red marked arrow. The green dotted border encloses a model as it is at a certain moment in time. The red dotted border encloses an updated version of this model. The green/red marked arrow indicates the evolution of one model to another model. This evolution between two models is called an *increment,* which refers to the changes in the models during a method adaptation.

1.3 Research design

The following sections elaborate on the research objective, the (sub)research question(s) and the utilized approach in this research. This section provides a more profound understanding of the way this research was carried out and with what rationale.

1.3.1 Research objective

In this research, we aim to incrementally change the *model level* when a *meta-model level* adaptation occurs. For instance, we have a model in the form of a Process Deliverable Diagram (PDD). Based on its abstraction level, this model resides at the meta-model level (marked with a green border in Figure 1). The PDD is a meta-model of the model that is operationalized on the model level. Both levels conform to each other (indicated by an orange arrow in Figure 1). When a user updates the model on the meta-model level (and thus creating a new meta-model [marked with a red border in Figure 1]), the model level automatically changes to fit the new meta-model level. In this way, the process model at the meta-model level automatically defines the interface of the software engineering tool at the model level.

The transformation process between the meta-model level and the model level, which in this research is referred to as *incremental method enactment*, will be subject to this research. The term *incremental method enactment* refers to the automatic changes in software engineering tools according to incremental changes in method increments.

1.3.2 Research question

The following main research question is defined for this research:

"How can method increments be employed in software engineering tools and allow for incremental method enactment?"

In order to answer the main research question, the main research question is divided in four sub questions:

RQ$_1$ *What are method increments?*

RQ$_2$ *Which adaptability requirements are available in software engineering tools?*

RQ$_3$ *How can data of method increments be used in software engineering tools?*

RQ$_4$ *When are product managers willing to adopt or use incremental method enactment?*

Each sub question addresses a part of the main research question. In the end, the answers to the sub questions will result in an answer to the main research question.

1.3.3 Research approach

This research focuses on the creation of an *IT artifact* that links the *meta-model level* with the *model level*. An IT artifact is defined as a bundle of "*material and cultural properties packaged in some socially recognizable form such as hardware and/or software*" (Orlikowski & Iacono, 2001, p. 121). It is typically a tailor-made solution for a specific scenario or problem.

The type of research for the successful completion of an IT artifact is defined as a design science research. A design science research attempts to create technology oriented solutions that serve human purposes (March & Smith, 1995). In existing literature we can find two highly appreciated design science research approaches, i.e. (1) the design science research according to Peffers, Tuunanen, Rothenberger and Chatterjee (2008) and (2) the design science research according to Hevner, March, Park and Ram (2004). Both models do not interfere with each other but rather complement each other. Yet in this research we prefer to use the approach as described by Peffers *et al.* as we believe this model has a more logical structure as opposed to Hevner's model. Also, the model of Peffers *et al.* is more recent and includes information as addressed by Hevner *et al.* Figure 2 depicts a graphical representation of the approach.

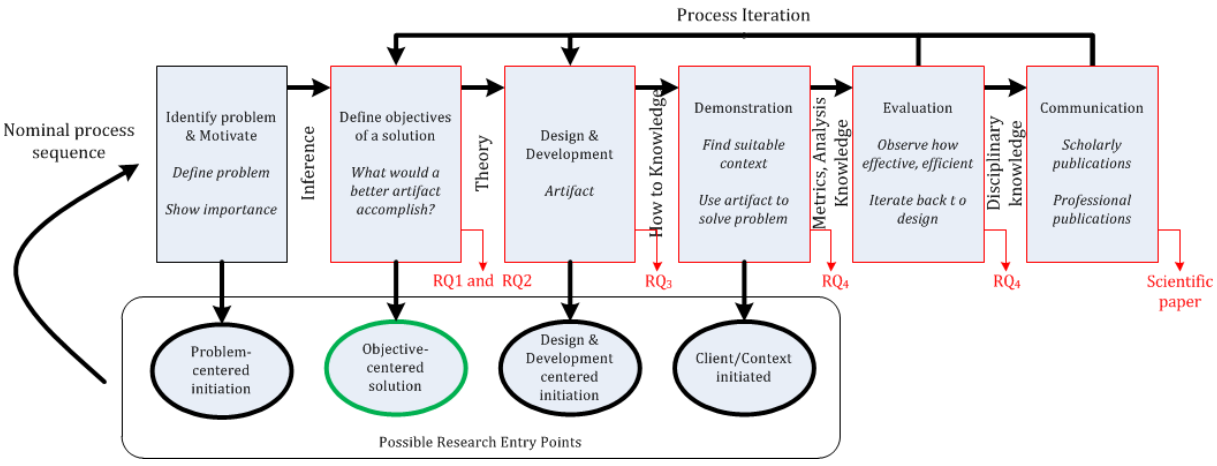


Figure 2. Design science research approach (Peffers *et al.*, 2008).

As we can see in Figure 2, a design science research can have different entry points. In total, four entry points are subject to a design science research. Each design science research can have a different entry point; however this does not mean that such a research differs from a research with a different entry point. The defined entry points are rather used to indicate at which step in the nominal process sequence you should start.

The entry point for this research is marked green in Figure 2, namely an objective-centered solution. The problem identification and motivation for our presented solution was also recognized by Niknafs and Ramsin in 2008. In their study they concluded that CAME environments have been remarkable but none of them actually provide a comprehensive set of means for enacting the method engineering process. Therefore this research proposes a solution to meet this concern by providing a means that supports the engineer in the enactment of a process.

The first sub research question in this research will be addressed in the define objectives of a solution step. To answer this sub question, we explore existing literature in the field of method engineering. We elaborate on the notion of method increments and on other concepts with which method increments are related. Furthermore, the second sub research question will also be addressed in this step. Based on literature research and internet research towards existing CAME and CASE tools, we first index software engineering and method engineering tools that are likely candidates for our research purpose. Based on the list of candidates, we select the most suitable and appropriate tool based on requirements that we derive from our architecture to method enactment. The requirements encompass features and components that the selected tool at least should have to achieve our research goal. Based on these criteria, we make a final selection for the tools that will be subject to this research. The results of the literature study and the defined requirements can be found in chapter 2 and chapter 3 of this thesis respectively.

After the tool selection, we address research question three with the rationale to design and develop the artifact. The artifact provides a means to support the engineer in the enactment of a method engineering process. The development of the artifact is partially based on existing theory, the objective and requirements that were addressed in the previous sub research question and our presented architecture to method enactment. An in-depth textual elaboration of the artifact will be provided in chapter 4. This chapter addresses the used techniques for the creation of the artifact, how it looks like and how it works.

After the elaboration of the artifact, we document our research findings with regard to incremental method enactment in chapter 5. In this chapter, we describe problems that arose during the enactment process. Furthermore, we also take expert evaluations into account with the aim to gather information about our presented approach and how it is perceived by the end-users. By conducting interviews with product managers (as they will be the end-users of our presented approach), we include opinions, remarks and other concerns and document these findings in chapter 5. Finally, this research closes with a discussion on incremental method enactment in chapter 6 and a conclusion on the results in chapter 7. Also, limitations and future work is included in the latter two chapters respectively.

1.4 Expert evaluation

The enactment mechanism that we create in this research will be evaluated by different product managers from different IT-companies. The rationale behind the expert evaluation is to elicit opinions, remarks and other concerns of the end-users (namely product managers) that will ultimately use the presented solution in this research. The derived information can be used as input for an updated version of the mechanism. Moreover, future work can be guided more specifically when taking the results of the expert evaluation into account.

The evaluation of our mechanism will take place in the form of a semi-structured interview. In this type of interview, some questions are prepared beforehand but there is also room for improvisation (Fontana & Frey, 2000). The reason for choosing a semi-structured interview is because we want to elicit opinions directly related to the mechanism itself. In addition, we also want to elicit general thoughts and remarks on the mechanism. Hence we also reserve time in which product managers can express their general feeling and general thoughts on the mechanism. For the evaluation of the mechanism, we aim to interview at least five different product managers from five different IT-companies as we believe this is deemed sufficient for a first evaluation of the mechanism. Each interviewed product manager and its respective company focuses on a different branch with its core product they are selling. In this way, we aim to gather a rather broad understanding of how the mechanism is perceived by different product managers in different IT branches.

The results of the interviews will be analysed twofold. In the first part, the answers from product managers related to our predefined questions will be conglomerated and merged to derive conclusions per question. Based on this information, we describe an overall conclusion on 1) the companies' current situation with regard to process adaptation and process evolution and on 2) how our presented enactment mechanism is perceived by different product managers. In the second part, we merge and combine data related to general opinions and remarks of product managers on the mechanism. This data will be collected and textually described as important factors that should be taken into account while developing a newer version of the mechanism.

1.5 Scientific contribution

In existing literature, there is hardly information available with respect to the operationalization of a (situational) method. However, there is a lot of literature available on the creation of situational methods and how these situational methods can be used to support the software engineering process. In 1996, MetaCase made the first attempt with MetaEdit+ to integrate the CAME and CASE environment by providing a tool that enables the creation of (situational) methods through domain specific modelling (referred to as the MetaEdit+ *workbench*). The developed method could consequently be operationalized with its respective CASE tool features (referred to as the MetaEdit+ *modeller*).

Still, the operationalization of the developed meta-model is just another approach for the creation of (situational) methods. Although these methods can be made domain specific and tailor-made to suit the needs to a particular situation, they do not add to the existing problem that methods usually do not conform to the actual working situation. A main cause for this problem is that a developed method cannot be further decomposed in an already employed software engineering tool. Therefore, conformance between the models and actual working situations are entirely dependent on manual intervention.

The research in this thesis takes the development of the (situational) method one step further by decomposing the developed method into an already employed software engineering tool. By not only focussing on the processes that indicate how the method should be carried out, we also emphasize the importance of the deliverables that are being produced while carrying out these processes. By validating our proposed solution and demonstrating it through a proof of concept, we show the feasibility, applicability and advantages of conforming two environments in order to keep the process models in accordance with the employed software engineering tool.

CHAPTER 2

Theoretical background

Every organization or sector that is concerned with software development employs Information Technology, regardless of its size. We can certainly assume that the development of software engineering projects is usually a difficult and complex task. A typical software development process involves many developers that participate in the development process by using a wide variety of development notations and tools such as the Unified Modelling Language (UML) or Computer Aided Software Engineering (CASE) tools. This chapter will introduce existing literature related to the focus of this research.

2.1 Software development

The development of information systems rapidly increases. Inevitably, the complexity and diversity of its application increases at an equivalent pace. To overcome these concerns, different methods (e.g. Demarco, 1979; Rumbaugh, Blaha, Premerlani, Eddy, & Lonrensen, 1991; Jacobson, Booch, & Rumbaugh, 1999) for developing Information Systems have been introduced. Information System Development Methods are collection of procedures, tools and documentation aids which will help the system developers in their effort to implement a new information system (Avison, 1988, as cited in Harmsen, Brinkkemper, & Oei, 1994). Information System development methods are the most significant key factors to achieve great success in development projects (Brinkkemper, Saeki, & Harmsen, 1999).

A survey among 132 organizations from Russo, Wynekoop and Walz (1995) revealed that 65% of the companies developed (or adapted) their used methods in-house. Putting methods effectively in practice requires hence a lot of time and effort. One of the causes for this problem is that existing Information System development methods are often too generic and cannot be followed literally (Harmsen *et al.*, 1994; Arni-Bloch, 2005). They need to be tuned to a specific situation for the project at hand (Harmsen & Brinkkemper, 1995; Brinkkemper, 1996; Rolland, 1997; Saeki, 2003) by incorporating the uniqueness of a specific project (Kumar & Welke, 1992). In contrast, 89% of the respondents in the study carried out by Russo *et al.* (1995) also agreed upon the fact that the utilized methods should be adapted to a specific project situation.

One of the biggest advantages of tailoring methods to a specific situation or domain is that it enables the creation of valid diagrams with the use of a simple user interface. Domain-specific modelling allows the engineer to add custom rules and constraints to a diagram. If a user would create a diagram in a universal software development tool such as Microsoft Visio, the user is not constrained by the actual modelling rules or notations of the technique he or she is using. As such the user would rather quickly develop a diagram that is unreliable and by no means adheres to the standards of the modelling technique. Certain diagrams become easily useless when they are applied in automated development processes as they do not reflect its standards.

2.2 Method Engineering

The field in which methods are constructed is the field of *Method Engineering*. Related to the Information Systems domain, Method Engineering focuses on the construction of new methods based on (parts of) previous methods and is defined as the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems (Brinkkemper, 1996). In order to develop meaningful methods, a structured procedure and representation is required (Brinkkemper *et al.,* 1999) because it would make no sense to construct a method lacking any meaning. Figure 3 illustrates the entire method engineering process.

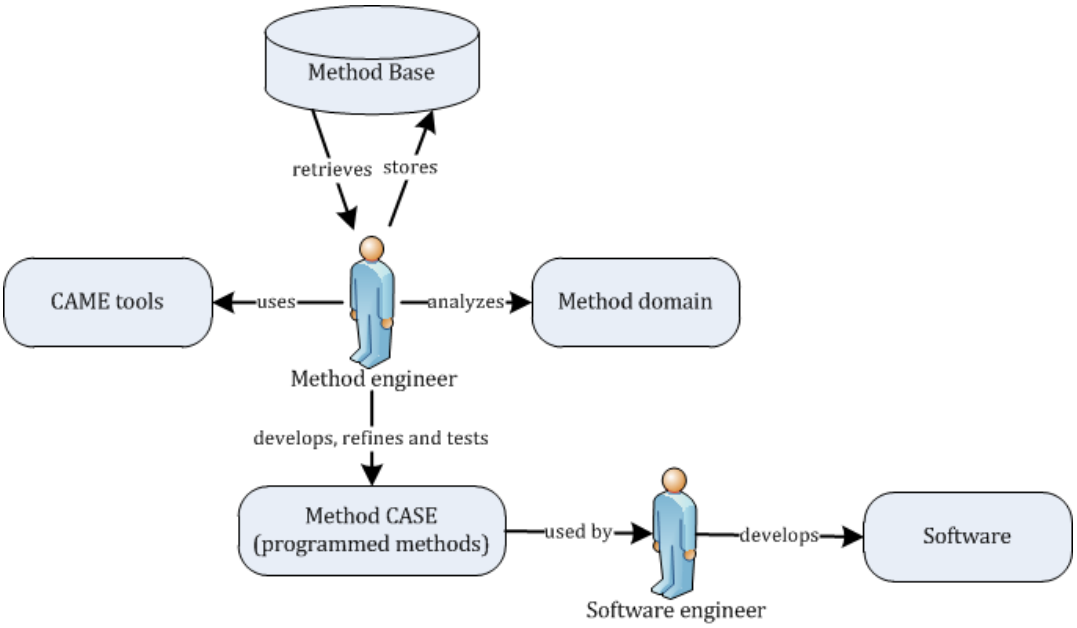


Figure 3. Method Engineering Process according to Saeki (1998) and Tekinerdoğan (2000).

At the heart of the image, the *method engineer* is shown. The method engineer is concerned with all the tasks related to the development of a method, for instance a method that can be used in the situation he or she is addressing. In doing that, the *method domain* describes what the method engineering should analyse first before just simply constructing a method. Derived from this domain, usable parts of methods, tools or techniques can be identified and consequently be stored in the *method base.* The method base (sometimes referred to as *method repository* by e.g. Ralyté & Rolland, 2001) is a central repository that is used for the retrieval of usable parts of a method. These parts can be used to assemble a specific method.

Engineering methods can be supported by using *Computer Aided Method Engineering (CAME) tools*. CAME tools support the method engineer in constructing methods by retrieving and, if desired, adapting existing parts of a method that were retrieved from the method base. This results in a constructed (situational) method, used by the *software engineer.* The software engineer utilizes *Computer Aided Software Engineering (CASE) tools* for the development of *software* based on guidelines that are attached to the assembled method by the method engineer.

2.2.1 Situational method engineering

The effectiveness of a particular method can be improved by tailoring a method to a specific situation. This type of engineering is referred to as *Situational Method Engineering (SME).* SME focuses on the configuration of system development methods that are tuned to the situation of a project at hand (Harmsen & Brinkkemper, 1995).

Since the introduction of the concept of SME, the adoption of SME highly increased and various approaches have been introduced in literature (e.g. Brinkkemper, 1996; Ralyté, Deneckère, & Rolland, 2003 and Weerd, Brinkkemper, Souer, & Versendaal, 2006). Situational Method Engineering is closely related to Method Engineering in the sense that Method Engineering is not concerned with situational factors. Engineering situational methods requires standardization of building blocks and guidelines in order to assemble a meaningful method. The building blocks of a method are called method fragments (Harmsen *et al.,* 1994; Brinkkemper, 1996). In Figure 4, the process of situational method engineering according to Brinkkemper (1996) and Harmsen (1997) is illustrated.

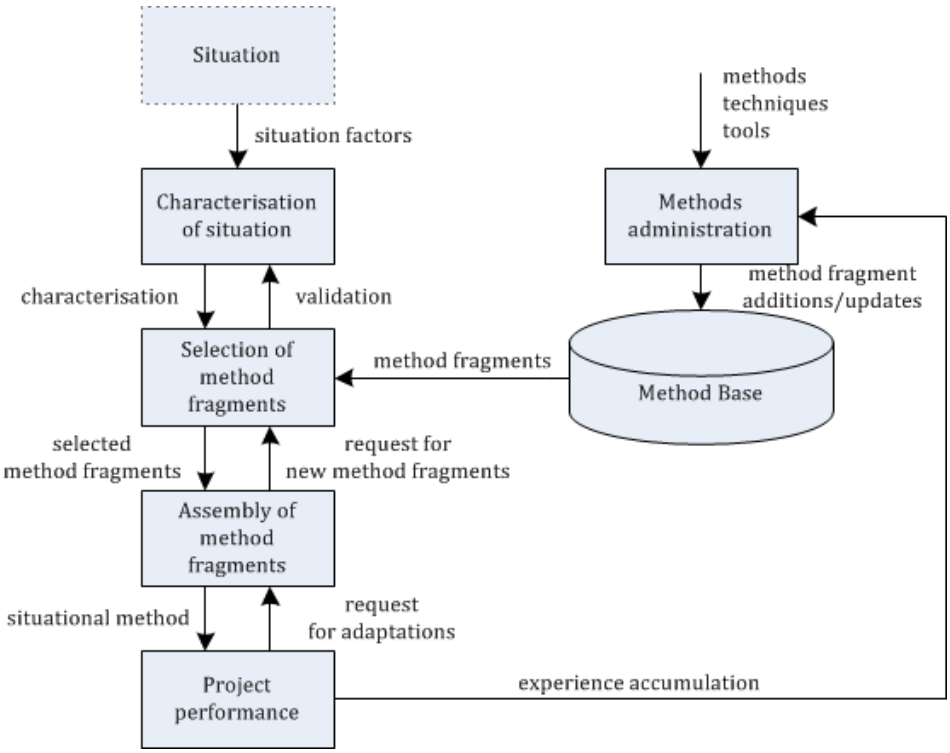


Figure 4. Process of situational method engineering (Brinkkemper, 1996; Harmsen, 1997).

As illustrated in Figure 4, the process for situational method engineering implicates an iterative approach and should therefore be interpreted with care. The starting point for developing a situational method is a specific (project) situation. A specific situation has *situational factors* that are characterized using a set of contingencies or contingency factors. Contingency factors, for instance contextual or technical factors, are assigned a value by the method engineer that determines the specific situation he or she is dealing with. When the situation is properly defined, the appropriate method fragments (or method chunks) that are stored in the method base can be retrieved. The *selection of the right method fragments* is based on the determined (project) situation and the guidelines and rules that are attached to the method fragments. For

example, if the method engineer discovers similarities with prior projects, the engineer can select method fragments that were beneficial in that specific scenario.

After the selection of the appropriate method fragments, the method fragments should be assembled in a consistent and sensible manner. By taking the guidelines of the method fragments into account, the method engineer should come up with a situational method that is flexible, suitable, consistent, efficient and robust. To success in this complex task, the engineer should follow the assembly rules that are attached to the method fragments. Also, the engineer could include a review step during the development of the situational method before he or she proceeds.

When the situational method is created, the actual use of the situational method is measured by its performance. Measuring the performance of a situational method does not differ from measuring the performance of conventional methods, except for the higher degree of flexibility (Harmsen, 1997). The performance of the situational method can trigger the process of adapting their current method. If the consistency and completeness require additional method fragments, the process of selecting and validating the method fragments is repeated until the situational method is deemed sufficient. The accumulated knowledge from this process is stored in the methods administration function that causes additions or updates to existing method fragments.

The *method base* (or method repository) is showed on the right hand side of the diagram together with its methods administration function. In the method base, *method fragments* (or method chunks) are stored that may have been collected from previous methods, tools or techniques. The methods administration function is intended to capture and provide methodological knowledge about method fragments.

2.2.2 Method fragments

The term *method fragment* was introduced by Harmsen *et al.,* (1994) and Brinkkemper (1996) later defined it as coherent pieces of Information System development methods. Hence method fragments can be regarded as an atomic element of a method (Henderson-Sellers & Ralyté, 2010).

In situational method engineering, two types of method fragments exist, namely (1) process fragments and (2) product fragments. The process fragments are models of the development process that visualize which steps need to be carried out and by whom. The product fragments encompass the structure of the products (e.g. models, tables, diagrams) that are being produced during the development of a situational method.

Some authors (e.g. Rolland, Plihon, & Ralyté, 1998; Ralyté, 1999 and Kornyshova, Deneckère, & Salinesi, 2007) introduced the term *method chunk* to emphasize the more constructive collection notion (Henderson-Sellers, Gonzalez-Perez, & Ralyté, 2008). By definition, a chunk is *"one process-focussed fragment plus one product-focussed fragment"* (Henderson-Sellers *et al.,* 2008, p. 483). Unlike the different naming convention and the slightly different approach between both notions, both approaches perform equally well in terms of capturing situational information (Henderson-Sellers *et al.,* 2008). For the sake of consistency throughout this research we prefer to use the term method fragment.

2.2.3 Process Deliverable Diagrams

An approach especially developed for method engineering purposes is the *Process Deliverable Diagram* (PDD) (Weerd & Brinkkemper, 2008). Process Deliverable Diagrams is built by

assembling method fragments from the method base. A PDD comprises two views, both integrated in one diagram. The first view is the process view of the diagram that reveals the relations between activities carried out by a user. The second view of the diagram is the deliverable view and reveals the deliverables produced in the process. The deliverable view is based on the notion of UML (Object Management Group, 2004) and hence depicts a similar representation. Also, a PDD is bound by different rules and constraints that should be taken into account while developing a method. Figure 5 illustrates an example PDD.

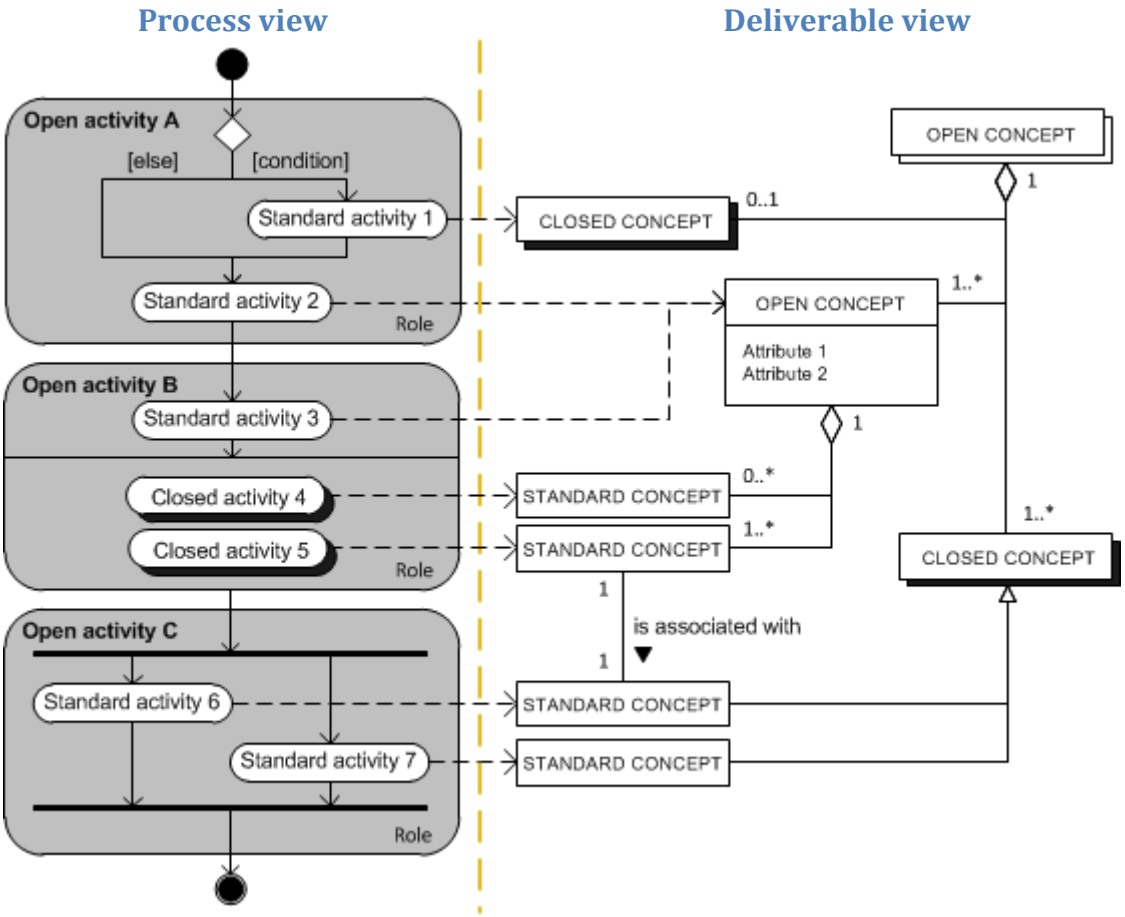


Figure 5. Example of a Process Deliverable Diagram.

In Figure 5, the distinction of the two types of method fragments that we mentioned earlier in the previous section is clearly visible, i.e. a process fragment and a product fragment. A process fragment (for instance an activity) is modelled on the process view-side of the PDD. A product fragment (for instance a concept) is modelled on the deliverable view-side of the PDD.

With respect to the left side of the diagram, a start and a stop fragment is in place. In between, three types of activities exist, namely a standard activity, an open activity and a closed activity. A standard activity contains no further (sub) activities. An open activity contains (sub) activities that are known and/or relevant in the specific context. A closed activity contains (sub) activities that are not known or not relevant in the specific context. In addition to the three types of activities, three possible scenarios can occur. Activities can be carried out (1) sequentially (e.g. from top to bottom), (2) unordered (e.g. from bottom to top or in any other order), or (3)

concurrently (e.g. simultaneously according to a predefined order).When decisions need to be made during a process, a condition determines the subsequent activities. Conditional activities are activities that are only carried out if a predefined condition is met (in other words: a Boolean clause). Conditions are represented using a branch (see Figure 3) and are denoted by two brackets: "[ ]".

Looking at the deliverable view of a PDD, three main types of concepts are distinguished, namely: (1) a standard concept, (2) an open concept and (3) a closed concept. A concept is a set of objects that share the same attributes, operations, relations and semantics (Booch, Rumbaugh, & Jacobson, 1999). Related to PDDs, a concept is any deliverable that is being produced when a certain activity is carried out.

A standard concept is a concept that contains no further concepts and is visualized by a rectangle. An open concept is similar to a standard concept but has a white shadowed border next to it. An open concept is used when the aggregated structure is shown in the same diagram or in a separate diagram. Finally, a closed concept is visualized using a black shadowed border contrary to an open concept. A closed concept is used when a concept has further concepts that are not know or not relevant in the specific context. Also note that every concept can contain attributes/properties. However, since this is not always the case, they may have been omitted from the diagram. In addition, and similar to the notion of UML, PDDs make use of multiplicities, aggregations, generalizations and associations.

When a situational method is altered or updated, a *method increment* is produced.

2.2.4 Method increments

A method increment is a collection of method fragments that have been inserted, modified or deleted during a certain method adaptation (Weerd, Brinkkemper, & Versendaal, 2010). Figure 6 shows an example of a method increment with the use of an evolutionary approach called incremental method evolution. Incremental method evolution is evolutionary approach to the maturation of method fragments. In the increments in Figure 6, coloured overlays are used to indicate the evolution of the increment. The meaning of these colours is as follows:

A method fragment or part thereof is inserted or modified.

A method fragment or part thereof is deleted.

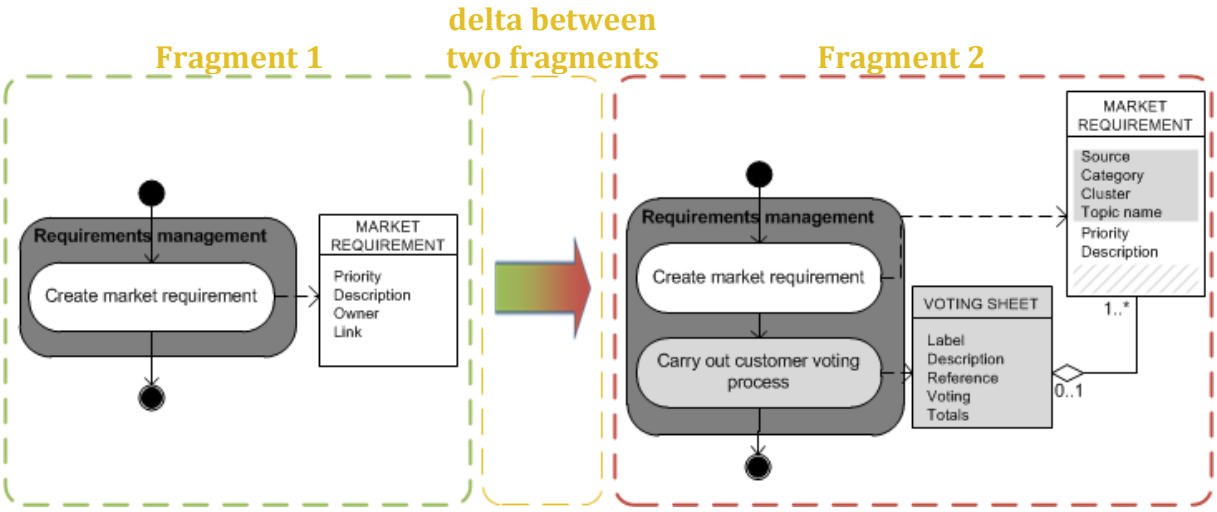


Figure 6. Example of a method increment in an evolutionary setting.

In Figure 6, two snapshots (in this research referred to as *fragments*) are provided in the form of a simple PDD. A *snapshot* is a model of a process as it was at a certain moment in time (Weerd, Brinkkemper, & Versendaal, 2007). In fragment 1, a rather simple requirements management process shows how a MARKET REQUIREMENT is stored and with which attributes. In fragment 2, the same requirements management process is provided, however it has been updated to suit the needs of a particular (project) situation better. The difference between two fragments, i.e. the delta between the fragments, is referred to as a *method increment*.

As visible in Figure 6, the MARKET REQUIREMENT concept has been updated with new attributes such as a source, category, cluster and topic name. The insertion or addition of these attributes is marked with a grey background colour. In addition, the owner and link attribute are removed (marked with a grey striped overlay). It is to note that a method adaptation only occurs if it the overall performance of the method will increase (Weerd *et al.,* 2007). Furthermore it is visible that a new process fragment called '*Carry out customer voting process'* has been inserted. Along with this activity, a VOTING SHEET with its accompanying attributes is required in order to document the process' results.

In total, 18 elementary increment types can be identified (Weerd *et al.,* 2007), namely:

- *Insertion* of a concept, property, relationship, activity node, transition, role.
- *Modification* of a concept, property, relationship, activity node, transition, role.
- *Deletion* of a concept, property, relationship, activity node, transition, role.

Based on these 18 increment types, an existing method increment can evolve from a very simple method towards a very comprehensive method and the other way around.

2.3 Engineering environments

In this research, two types of environments are subject to investigation, namely the Computer Aided Method Engineering (CAME) environment and the Computer Aided Software Engineering (CASE) environment. The following subsections will introduce both concepts along with an overview of existing tools in their domain.

2.3.1 CAME environment

A Computer Aided Method Engineering (CAME) tool is a kind of computerized tool that supports the process of creating situational methods (Saeki, 2003) with the use of a repository that stores the reusable method fragments. A CAME environment enables the development of a flexible support environment for information engineers that are engaged with analysis and design activities (Dahanayake, 2001). Usually a set of correlated tools is in place to emphasize re-use.

In literature, CAME environments receive a rather amount of attention. Still, very few CAME tools have been introduced until today. The CAME tools that have been introduced are rather simple academic prototypes than more advanced engineering tools. Strangely there is only one remarkable and comprehensive tool available in this domain called MetaEdit+. Nonetheless, Table 1 provides an overview of previously introduced CAME tools.

Table 1. Overview of previous introduced CAME tools.

CAME tool	SME approach	Environment usage
MERET (Heym & Osterle, 1992)	Method customization	Research
MethodBase (Saeki, Iguchi, Wenyin, & Shinohara, 1993)	Method customization	Research
Decamerone (Harmsen & Brinkkemper, 1995)	Assembly-based	Research
MENTOR (Si-Said, Rolland, & Grosz, 1996)	Assembly-based, paradigm-based	Research
MetaEdit+ (Kelly, Lyytinen, & Rossi, 1996)	Assembly-based	Research and commercial
MERU (Gupta & Prakash, 2001)	Assembly-based	Research
Method Editor (Saeki, 2003)	Assembly-based	Research

As we can see in Table 1, only seven CAME tools could be found in literature. During the literature research we noticed that most of the found tools had a very concise description or were used in a specific research only. Therefore they were only rather superficial described and in most cases not even available for download. The only notable tool that is available for download and comes with a rather comprehensive description is the commercial tool called MetaEdit+. As a result, this tool is the most used in practice probably due to its comprehensive features and guidelines.

The approach for constructing situational methods with the use of a CAME tool can be accomplished in three ways, namely (1) *assembly-based*, (2) *extension-based* or (3) *paradigm-based* (Ralyté *et al.,* 2003). With the use of an assembly-based approach, a situational method is created by assembling and/or adapting method fragments that are stored in the method base; typically achieved with the use of a drag and drop approach via a user interface. Secondly, a situational method can be created by extending an existing method through the addition of (existing) method fragments. This approach is referred to as the *extension-based* approach as it extends an existing method. Thirdly, a situational method can be created by abstracting/instantiating an existing model/meta-model. This approach is called the *paradigm-based* approach.

As also drawn in Table 1, a fourth approach for the development of situational methods was introduced by Niknafs and Ramsin (2008), namely *method customization.* Method customization is the selection of a method and customizing it to fit the situation at hand. The repository only contains complete methods that can be altered and no (other) fragments. Even though each approach yields almost the same results, the assembly-based approach is generally used for the development of situational methods. The other approaches are completely overlooked and the process for assembling method fragments is still not tackled in literature (Ralyté & Rolland, 2001).

In 2008, Niknafs and Ramsin made a general analysis and comparison of the existing CAME environments. Table 2 summarizes these results.

Table 2. General analysis and comparison of the introduced CAME tools (Niknafs & Ramsin, 2008).

CAME tool	Number of features	Contributions	Available literature
MERET	Low	Average	Low
MethodBase	Low	Average	Low
Decamerone	High	High	Average
MENTOR	Average	Average	Average
MetaEdit+	High	Average	High
MERU	High	High	Low
Method Editor	Average	Average	Average

The most left column in Table 2 contains the previous introduced CAME tools. For each CAME tool, three types of criteria were investigated and assigned a *low/average/high* measurement. The distribution of the measurements was as follows:

- A *low* grade was assigned if the available publications on the CAME environment were rare or did not have sufficient level of detail.
- An *average* grade was assigned if more than one author had published a paper on the CAME environment with the exception that the paper differed from the level of detail compared to the original paper.
- A *high* grade was assigned if more than one author published more than one paper on the CAME environment at different levels of detail.

Based on the results drawn in Table 2, we can see that MetaEdit+ and Decamerone are the only available CAME tools that provide sufficient amounts of features and literature to based further research upon. The other available tools are too superficial and too specific to a particular research environment.

2.3.2 CASE environment

A Computer Aided Software Engineering (CASE) environment is an environment or a set of tools that support software engineering methodologies and aid engineers in automating their software development (McLure, 1989). Hence a CASE environment usually consists of multiple (employed) CASE tools. A CASE tool is a computer-based product aimed at supporting one or more techniques during the development of software (Jarzabek & Huang, 1998) and often varies widely in scope. Were some of the CASE tools are simple diagramming tools, other tools can vary to comprehensive integrated solutions that aims to facilitate all phases of the system development lifecycle (Alavi, 1993). However, using different types of CASE tools can usually

make the integration of individual tools and the coordination with project team member's difficult (Premkumar & Potter, 1995),

CASE tools are often partitioned in two types of CASE tools, namely *upper CASE* and *lower CASE* tools (Wasserman, 1990). Upper CASE refers to the support of the analysis and design of a software system and lower CASE refers to the support of coding and testing of a software system. However, in that same research Wasserman also concludes that this distinction is not really necessary when someone tries to integrate both tools. Hence we will not use this terminology throughout this research.

In literature, CASE tools received a lot of attention. At the same time however, the definition to define a CASE tool was often only used until the end of the 90-ties and from that point, little CASE tools are described in literature. CASE tools were probably of great interest of researchers in that period and it was probably also hard to distinguish between different tools due to the extremely increased amount of software being produced in that period. As a result, we see that almost every software engineering tool (i.e. a tool that supports the engineering of software) is referred to as a CASE tool today. In fact, even a simple class that is written in any programming language is already referred to as a CASE tool.

Because almost every simple software engineering tool is either referred to as a CASE tool or is just referred to as software related to its domain, an extensive overview of existing CASE tools is impossible to realize as this would be gigantic. Yet, an attempt to create an overview is provided on the website case-tools.org[1]. The website lists a comprehensive overview of different tools in different domains. Because we are focusing on the prioritization of requirements, we narrowed down our focus towards CASE tools that provide capabilities to support requirements prioritization. Note that we only selected those CASE tools that are rather comprehensive by default because in that case we are sure that such a tool contains sufficient features to make it a suitable candidate for our research purpose. Hence we eliminated all the tools that were too superficial or too concise because these were usually mock-ups. Table 3 provides an overview of the selected CASE tools.

Table 3. Overview of existing CASE tools focusing on the prioritization of requirements.

CASE tool	Website
Accept requirements management	http://www.accept360.com/how-we-help/solutions/requirements-management
IBM DOORS	http://www-01.ibm.com/software/awdtools/doors
IBM RequisitePro	http://www-01.ibm.com/software/awdtools/reqpro
Atlassian Jira	http://www.atlassian.com/software/jira/overview
Borland CaliberRM	http://www.borland.com
Micro Focus Optimal Trace	http://www.microfocus.com/products
Open Source Requirements Management Tool (OSRMT)	http://ostatic.com/osrmt
Accompa	http://www.accompa.com

Based on the selected CASE tools that are summed up in Table 3, each tool is a likely candidate for this research purpose.

[1] http://case-tools.org

2.4 Integrating the engineering environments

In order to maintain consistency between the CAME and CASE tool, an artifact needs to be created. The development a complex artifact can be difficult but yet is a vital and important part. Good modelling techniques offer support for the separation of concerns principle, analysis of designs and for structuring construction activities (France & Rumpe, 2005). Therefore we need a number of computer-based design tools that support us in the development of complex engineering artifacts (Karsai, Lang, & Neema, 2005). However, employing all kind of design tools does not necessarily enhance this process. If we would use multiple tools throughout this research, the data would be distributed all over these tools. For that reason it is necessary to make relationships of different tool data repositories visible and keep them consistent with each other (Könings & Schürr, 2006). We need to be aware of the relationships between semantic equivalent data in different repositories in order to preserve consistency, maintainability and traceability (Amelunxen, Klar, Königs, Rötschke, & Schürr, 2008). A solution to this problem could be found by using a mapping that maps the data that is in both repositories, thus providing a means to preserve consistency, maintainability and traceability.

Exchanging the data of models among different tools is an important prerequisite for achieving effective software development as tools are typically produced by different vendors where each vendor chooses to focus on a specific part of the overall problem (Wasserman, 1990). This is a reason why tool integration has been recognized as a key issue in complex, computer-supported engineering processes (Karsai, Lang, & Neema, 2003). Integrating tools is recognized as a difficult, labour intensive activity (Karsai *et al.,* 2003) and the lack of interoperability between different tools makes it hard to use these different tools in combination (Kramler *et al.*, 2006).

To aid software engineers in this problem, organizations can employ an integration platform such as the Eclipse Environment[2] or IBM Rational Rose[3] package. These platforms congregate the necessary tools in one package thus reducing the distribution of data throughout multiple tools. Also, the amount of different tools from different vendors is substantially diminished. However, if a development process includes ingredients that are not supported by the elements of the platform, the engineer faces the tool integration problem again (Karsai *et al.,* 2003). As such these "integrated" platforms easily become inconsistent and unusable due to overlap and redundancy in the data (Burmester e*t al.*, 2004).

2.5 Relation to Business Process Management

The goal of this research has similarities with work that has already been executed in the field of Business Process Management (BPM). BPM also includes methods, techniques and tools to support the design, enactment, management and analysis of operational business processes (Aalst, Hofstede, & Weske, 2003). The research we are carrying out, in contrast to the field of BPM, fundamentally differs from the field of BPM because we focus on the development of software; a process that is typically iterative, unpredictable and influenced by a lot of external factors. BPM however, is used to guide (business) processes in a standardized and repetitive manner (for instance invoicing). In certain processes, typically no external influences exist and

[2] http://www.eclipse.org
[3] http://www-01.ibm.com/software/awdtools/developer/rose

the respective process can be carried out according to a predefined process model. Also, the research that we are addressing in this study does not focus only on the activities within a (business) process, but also emphasizes the importance of the deliverables that are being produced while a certain activity is carried out. Because BPM focuses rather on the activities that need to be carried out, the focus on the results of that activity is typically lacking. It is rather a tool to standardize and guide (business) processes with a predefined sequence and therefore BPM is not suitable for an environment where a process is iterative, unpredictable and influenced by (a lot of) external factors.

2.6 Terminology explanation

In this research, we use different terminology that has a specific meaning in the context of this research. In order to prevent fragmentation of the meaning of the used terminology, we elaborate on the most important concepts in this research in Table 4 below.

Table 4. Terminology in the context of this research.

Term	Context in this research
(Business) process	A set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization but it may interact with business processes performed by other organizations (Weske, 2007).
CAME tool	A kind of computerized tool that supports the process of creating situational methods (Saeki, 2003).
CASE tool	A computer-based product aimed at supporting one or more techniques during the development of software (Jarzabek & Huang, 1998).
Conformance [between two (architectural) levels]	The degree to which a model conforms to its respective meta-model.
Enactment	The automatic changes in a software engineering tool based on a method increment.
Incremental method	The evolution of a method over time.
Mechanism	A bunch of programming classes that enable the change in a software engineering tool.
Meta-model	An abstract model of a model.
Method	An approach to perform a system development project, based on a specific way of thinking, consisting of directions and rules, structured is a systematic way (Brinkkemper, 1996).
Software Engineering Tool	*See CASE tool.*
Situational method	A method that is tailored to a specific situation.

CHAPTER 3

Adaptability Requirements of Software Engineering Tools

The most important decision that needs to be made during this research is the selection of the right software engineering tools that allow for incremental method enactment. Choosing the wrong tool will affect the entire feasibility of this research and its results with regard to method enactment. Therefore, this chapter explains the approach to incremental method enactment followed by the requirements that the CAME tool, the CASE tool and the mechanism needs to address. Based on the requirements, the CAME tool, the CASE tool and the enactment mechanism will be selected. After the selection, we draw the architecture for method enactment. Finally, the chapter closes with the characteristics of the selected CAME tool CASE tool to derive how their available features can be of value for the remainder of this research.

3.1 Approach to incremental method enactment

As we also illustrated in Figure 1, we aim to maintain consistency between two architectural levels. In order to realize this, a more profound understanding of the approach is used. Figure 7 depicts a graphical representation of the approach in more detail.

As we can see in Figure 7, the four architectural levels as defined by the Object Management Group (2011) are depicted. These levels are in accordance with Figure 1. To increase visibility however, the roles are here omitted from the diagram. The left-hand side of Figure 7 shows the CAME tool (marked in blue) and the right-hand side of the diagram shows the CASE tool (marked in grey).

At the *meta-meta model level*, we find the data that is stored in the repository of the CAME tool. The structure of this data is based on the meta-meta model and it provides insight in how the data at the *meta-model level* is structured. The data that is stored in the repository can by no means deviate from its meta-meta model as they are always in accordance with each other.

On the *meta-model level,* the repository of the CAME tool and CASE tool is situated. The repository stores all the data that is entered through the respective tool, either graphical or textual. With respect to the CAME tool, the repository contains the data of the diagrams that are created with the tool itself and the meta-model of the respective diagrams. With regard to the CASE tool, the repository stores the data that is entered in the CASE tool based on its meta-model. In case of the CASE tool, the repository is a relational database so that data is stored in an efficient and unambiguous manner.

On the *model level,* the CAME tool is equipped with a generator. A generator is a functionality of the CAME tool that provides the user with a possibility to add additional functionality. By default a CAME tool comes with different features such as the exportation or conversion of a diagram. If desired, the generator can be used to add supplementary functionality to the tool to use and modify the data that is stored in the repository. With regard to the CASE tool,

administrative functions are usually in place to modify the tools visibility, functionality or to make changes to the data in the repository on the meta-model level.

Finally, the *data layer level* is the level a regular user is concerned with and is the Graphical User Interface (GUI). This level is only relevant for users that are using the CAME tool or CASE tool to enter data to the repository, e.g. by adding a new record to the database through the GUI.

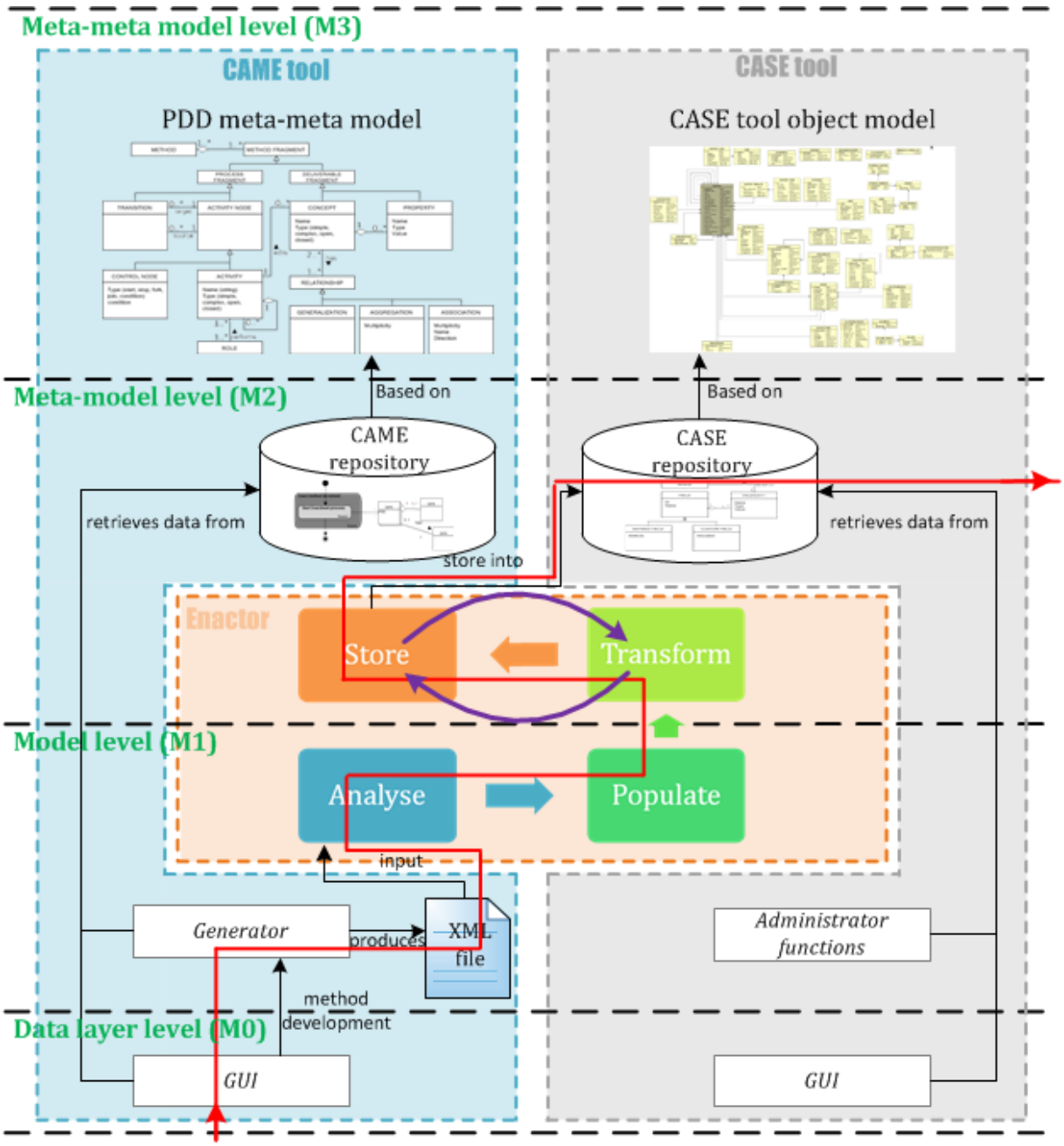


Figure 7. Approach to achieving incremental method enactment.

The red line in Figure 7 shows the process for enacting a fragment incrementally. The process starts at the data layer level and ends at the meta-model level. In between, the enactment mechanism is shown, constituting four phases, namely *analyse, populate, transform* and *store.*

Data layer level (M0)

The entire enactment process is trigged by the completion of creating a valid Process Deliverable Diagram (referred to as a fragment in this research). The fragment is created with the use of the Graphical User Interface (GUI) of the CAME tool and is stored in the CAME tool's *repository* for future usage. When the fragment in completed, the *generator* can be invoked.

Model level (M1)

After the development of a valid fragment, the current fragment someone is viewing will be exported to plain text through the use of a *generator.* The type of text that we want to use for this file is the uniform language format called eXtensible Markup Language (XML) (Bray *et al.*, 2006). The reason for choosing upon XML is because XML diminishes obstacles to sharing data between diverse applications and database (Seligman & Roenthal, 2001). The format is easy to understand to both humans and machine. As such it has become the standard technology for software engineers (Salminen & Tompa, 2011). When the current fragment is exported to the XML language format, the XML file is analysed by the *enactment mechanism* in order to determine whether the XML file is valid and originated from the CAME tool. If this is the case, the XML file is read. Because the data of this XML file is surrounded by different types of XML notations and most likely data that was sent from the CAME tool, the enactment mechanism needs to know how it can derive and temporary store the right data from the XML file in a sound manner. To accomplish this, we base our solution on the PDD meta-meta model (Weerd *et al.,* 2007). Based on this model, the enactment mechanism will analyse the data within the XML file and derive the required data for this research. While doing this, the enactment mechanism will populate the data from the XML file to a new data format that is in accordance with the PDD meta-meta-model. This process will continue until the whole XML file is read. Any other data within the XML file that is of no use will be neglected.

Meta-model level (M2)

When the data is populated in accordance with the PDD meta-meta model, the data needs to be transformed to a data format that is in compliance with the CASE tool. Because the enactment mechanism needs to know how the data from the PDD meta-meta model can be converted to the CASE tool object model, the object model of the CASE tool is needed. Based on the object model of the CASE tool, the data will be transformed to fit this model. When this transformation process is completed, the data needs to be stored in the CASE tool's repository. However, before any data can be stored in to the repository of the CASE tool, the data needs to be placed in Structured Query Language (SQL) queries first before they can be understood by the repository. When the SQL queries are created, the queries are executed whereby they will store the temporary data definitive in the repository of the CASE tool.

Based on the described approach, we can see that the enactment mechanism is only concerned with the *meta-model level* and the *model level*. The *meta-meta model level* is already defined by the CAME tool and the CASE tool and the *data layer level* is only used to represent the data that is stored within the CAME tool and CASE tool in a logical and understandable manner to humans. In chapter 4 we will elaborate extensively on the enactment mechanism itself.

3.2 Selection of the CAME tool

As we described in section 2.3.1, seven CAME tools were found in existing literature. MetaEdit+ is the only CAME tool that is being used for commercial purposes as well as scientific purposes. The other CAME tools are used for scientific purposes solely. A disadvantage of tools that are used for scientific purposes is often the fact that they are usually tailor-made to a specific scenario and thus are not generalizable to an entire domain. Certain tools are often created for a particular scenario that was subject to the research the authors were addressing. This makes the tool to be hardly useful in any other experimental setting and often very limited in functionality. Even though this does not necessarily mean that these tools are useless in any other experimental setting, we also found that the tools were not even available for download. The only notable tool that was available for download was the tool called MetaEdit+.

3.2.1 CAME tool requirements

Even though MetaEdit+ remained as a likely candidate for this research, it needs to include the right functionality needed for this research. In order to incrementally enact a fragment from the CAME tool, the CAME tool should be capable of overcoming the following requirements:

1. **Diagram editor**
   The CAME should be able to create valid increments according to the notion of Process Deliverable Diagrams through a simple user interface.
2. **Plain text exporter**
   The CAME tool should be capable of transforming a graphical representation to structural textual data.
3. **Enactment mechanism launcher**
   The CAME tool should include a possibility to invoke the enactment mechanism that is responsible for maintaining the consistency between the meta-model level (i.e. the CAME tool itself) and the model level (i.e. the CASE tool).

Requirement 1: diagram editor

In 2010, Vlaanderen created method objects of a Process Deliverable Diagram (PDD) in MetaEdit+ with the use of the MetaEdit+ Reporting Language (MERL). In total, thirteen objects were created were each object represented a fragment of a PDD. With these thirteen objects, a complete PDD could be modelled. When you add an activity to your model for instance, you can add the 'activity object' to the model and fill in the necessary data of the object. Other fragments such as a start, stop, decision and open/closed concepts are available as well. In addition to these objects, the rules and constraints that apply to a PDD were partially taken into account to avoid the creation of an invalid diagram.

The aforementioned research of Vlaanderen addresses the first requirement for this research. With the objects available in MetaEdit+ we now can utilize MetaEdit+ as a tool that is capable of modelling a valid PDD. Even though this makes MetaEdit+ as a likely CAME tool candidate, the second and third requirement should not be overlooked since they are of equivalent importance in this research.

Requirement 2: plain text exporter

Because we aim to enact a fragment incrementally, the enactment mechanism should be capable of reading the fragment that we create and use that fragment as input for the CASE tool. Hence the CAME tool should be capable of transforming a graphical object (in this research a PDD) to plain text without losing the underlying integrity of the model. The best solution to this problem is adhering to a uniform language standard such as the XML file format. Fortunately MetaEdit+ includes an XML exportation option by default and bases its XML structure on the GOPRR conceptual data model (Kelly *et al.,* 1996). GOPRR is an abbreviation that stands for graphs, objects, properties, relationships and roles and it specifies how the data within the XML file is structured. Table 5 shows an example of the GOPRR data model when comparing it against the structure of a PDD.

Table 5. GOPRR data model in contrast to PDD fragments.

GOPRR entity	PDD example fragment
Graph	A fragment (i.e. a PDD)
Object	A concept or an activity
Property	The name of an activity/concept or an attribute of a concept
Relationship	A control flow, aggregation or generalization
Role	A link that connects an object to a relationship

A *graph* for instance indicates which type of graph the XML file belongs to. In this research, we use a fragment as our graph and therefore the graph would be a PDD. An *object* within the XML file is referred to as a concept or an activity that is modelled in the PDD. A *property* indicates the name of a method fragment or an attribute of a concept that is placed in the PDD. The *relationship* indicates which type of relationship connects different method fragments (for instance a generalization, aggregation or association) and a *role* associates an object entity to a relationship entity.

By mapping the GOPRR data structure against a PDD, we are able to distinguish between method fragments of a PDD and to associate them with an entity. This allows the enactment mechanism to distinguish different method fragments from each other and to rebuild the structure of the PDD based on the XML file.

Requirement 3: enactment mechanism launcher

As a last requirement, the CAME tool should include the possibility to invoke the enactment mechanism which is responsible for enacting the current fragment. With respect to this requirement, MetaEdit+ includes a feature that allows the user to add tailor-made functionality to the program. When certain functionalities are absent in the default package, a so called *generator* can be utilized that enables the user to add supplementary functionality to MetaEdit+. The generator in MetaEdit+ is basically a functionality that provides the user with a window in which you can write down certain commands; similarly to each programming language. In MetaEdit+, this default scripting language is the **Me**taEdit+ **R**eporting **L**anguage (MERL) and it allows the user to generate code or documentation for instance. By default, MERL comes with a rather comprehensive manual in which you can find a detailed description of the available functions.

3.2.2 CAME tool conclusion

By addressing the requirements for the CAME tool and providing solutions to overcome these requirements, we can conclude that MetaEdit+ is the most suitable candidate for this research. The tool is comprehensive and sufficient enough for this research purpose. In addition we have the PDD objects available in MetaEdit+ with which we can create valid PDDs through the MetaEdit+ interface. Moreover MetaEdit+ is able to transform a graphical fragment to structured text and has a possibility to invoke the enactment mechanism.

3.3 Selection of the CASE tool

The selection of the CASE tool is a much more complex decision compared to the selection of the CAME tool because there are a lot of CASE tools available. As we mentioned earlier, a lot of CASE tools can be found ranging from very simple object classes to very sophisticated supporting tools. Yet, we are constrained by the requirements and feasibility in this research and therefore the CASE tool should address some degree of flexibility with respect to its functionality.

3.3.1 CASE tool requirements

The *enactment mechanism* is responsible for maintaining the consistency between the CAME tool and the CASE tool. Because a change in the CAME tool results in an adaptation of the data within a CASE tool, the CASE tool should overcome the following requirements:

1. **Process side and deliverable side support**
   The CASE tool needs to include a means to support the process side of a PDD as well as the deliverable side of a PDD. Therefore the CASE tool should include a variety of functionality that allows the user to modify its visibility and behaviour.
2. **Well-known CASE tool**
   The CASE tool needs to be used in different organizations on a regular basis in various real-life settings (e.g. the CASE tool should be well-known to the IT-industry).
3. **External repository access**
   The data stored within the CASE tool should be accessible and modifiable without using the CASE tool itself.

Requirement 1: process side and deliverable side support

A Process Deliverable Diagram can be very dynamic and almost tuned to every situation. Because it is not known how a fragment will look like beforehand we demand the CASE tool to be as flexible as possible in contrast to the employed CAME tool. The CASE tool should be able to support the process flow that is present in the PDD and needs to incorporate the deliverables belonging to the flow. It is to say that finding such a tool is rather difficult because each tool has its own strengths, weaknesses and characteristics.

Requirement 2: well-known CASE tool

Obviously it would be useless to select a CASE tool that is not used within different organizations. Seldom used CASE tools are most likely tailor-made to suit the needs of a specific situation and they are often very limited in functionality and applicability. Selecting such tools would tremendously diminish the applicability and results of this research. Moreover we could

hardly generalize the results of such a CASE tool to an entire domain because it is simply too limited in functionality. As we mentioned earlier there are a lot of CASE tools that are in fact a bunch of classes containing lines of programming code. It is to say that these "tools" are very adaptable and easy to tune to this specific research goal. However, the applicability within organizations is very low and the way data is stored in certain tools is usually very poor. Moreover they are not used on a regular basis and often not used by a great amount of companies. This makes the results almost useless in any real-life setting.

Requirement 3: external repository access

The enactment mechanism is responsible for maintaining the consistency between the meta-model level and the model level, i.e. the consistency between the CAME tool and the CASE tool. For that reason we need to have external access to the data that is used within the CASE tool and we need to be able to modify this data without using the CASE tool itself. Every CASE tool that is in the cloud through your web browser is hence not an option because the data of such a CASE tool is hardly accessible. The most ideal solution to accessing data externally is through the use of a queried language such as the Structured Query Language (SQL). In such a situation, we can retrieve the appropriate data and change it accordingly if desired. Fortunately every respected CASE tool makes use of a central repository that stores the data in an efficient and understandable manner. Still, if we are not able to change the data externally, the whole enactment process would be impossible to realize.

3.3.2 CASE tool conclusion

The most important requirement the CASE tool needs to address is the fact that the tool needs to support the process side of a PDD as well as the deliverable side. Because the process side of a PDD is basically a workflow, the CASE tool should include a means to support this workflow. Due to the great amount of available CASE tools, we had to make a selection of usable CASE tools as likely candidates for this research.

For the selection of the right CASE tool, we first created a list of available CASE tools. The creation of the list was based on manual research on the internet and also included an overview of existing CASE tools that are listed on the website called case-tools.org[4]. This latter website indexes existing CASE tool per domain. In total, we found over 50 usable CASE tools.

Secondly, we narrowed down our list of CASE tools by using three types of criteria, namely

1. The CASE tool needs to be rather comprehensive by default and include a great amount of features and flexibility. CASE tools that were too superficial or too concise (such as small mock-ups or simple programming classes) were eliminated from the list.
2. The selected CASE should be well-known to the IT-industry. We marked a CASE tool as 'well-known' if we found numerous prominent companies (i.e. customers of the CASE tool) that are using the CASE tool.
3. The CASE tool preferably focuses on the prioritization of requirements because this research also focuses on this domain.

After applying the aforementioned criteria to our list of CASE tools, eight CASE tools remained. These eight CASE tools are also described in Table 3. Next, we looked at the fact which CASE tool

[4] http://case-tools.org

allowed us to access the repository of the CASE tool externally without using the CASE tool itself. During this process, we also looked at the available CASE tool features to find support for the process side of PDD as well as support for the deliverable side of a PDD. During this research, we stumbled upon the well-known project tracking tool Jira from Atlassian. We investigated its possibilities and noticed that Jira was a very good candidate for our research purpose. Jira supports workflow possibilities in which you can exactly reproduce the workflow that is present in the PDD. Furthermore it includes a rather great amount of flexibility and also a great variety of (administrator) functions. Moreover, we found a way in which we could map the features of Jira onto the method fragments of a PDD. This mapping will be explained later in chapter 4.

Based on the requirements we addressed in the previous section, we concluded that Jira was the most ideal candidate for our research purpose; especially if we compare it against the remaining CASE tools on our list. We do recognize that the selection of Jira may slightly affect the results of our research. Still, we aim to make our results generalizable to the entire CASE domain rather than making it specific to Jira only. Another issue that arises when using Jira as our CASE tool, is the fact that Jira uses the word *issues* to indicate a record. Although the naming convention implicates that data in Jira is defined as an issue, it is in fact an *entity* within Jira. In our research, this entity is a fragment in the form of a PDD.

3.4 Selection of the enactment mechanism

Finally, a decision needs to be made on the mechanism that is responsible for maintaining the consistency between the meta-model level and the model level. If we take a look at Jira and its respected features, a fragment can be imported in the repository of the CASE tool in three different ways:

1. Utilizing the Jira Application Programming Interface (API) for transferring data from and to the Jira repository.
2. With the use of the default import/export function of Jira.
3. Direct database access with the use of a Structured Query Language (SQL).

With regard to these solutions, using the Jira API is definitely the most secure solution with respect to mitigating any database related problems (such as pollution or incoherency between data). However, a disadvantage of using an API is that you are constrained by the features and possibilities of the API itself. Because we aim to enact a rather dynamic process, we do not want to be constrained by the available features. When we take a more profound look at the available features of the Jira API, we notice that we are not able to update all the required tables in the database. Some tables are only accessible by Jira itself or when looking directly into the database. The functionality of the Jira API is rather comprehensive by default but unfortunately not comprehensive enough to meet the requirements for this research. Therefore we had to eliminate this solution as a possibility for this research.

Secondly, we could choose upon the default importation/exportation feature of Jira. For instance, you export the current Jira data to an XML file and update that XML file with data that is elicited from the fragment. Consequently you import that XML file into Jira. However, it is to note that such a solution is by no means practical since you need to export the current Jira data to XML first. This also applies to importing the XML file into Jira. Although there might be a solution to overcome this issue, we noticed that importing a modified XML file repeatedly failed

because the data in the XML file was probably not aligned with the data in the database of Jira. This caused the importation process to be terminated before it had been finished leaving us no option to discard this solution as well.

Finally, we had the possibility to create a custom solution with the use of a programming language that inserts the data of a fragment into the repository of the CASE tool through a structured query language. Although this solution is the most flexible one as we can develop it as we want, we need to make sure that we exactly mimic Jira's behaviour to avoid pollution or inconsistency in the database. However, in this research this is not a major problem because Jira produces a log file when you add or change data in Jira. With the use of these log files, we can exactly reproduce Jira's behaviour by simply analysing this log file and performing multiple test rounds. Therefore, this latter solution will yield the most practical results with respect to the enactment of a fragment in Jira.

3.4.1 Enactment mechanism requirements

The tailor-made solution to incremental method enactment will constitute a bunch of commands that are written according to a particular programming language. The decision upon the programming language for the enactment mechanism is rather a preference of the programmer than a research need. Every respected programming language that is well-known to society and comprehensive enough will be sufficient for this research purpose. Still, the selected programming language should be capable of overcoming the following requirements:

1. **Reading XML**
   The selected programming language should be able to read an XML file that is produced by the CAME tool.
2. **Understanding SQL**
   The selected programming language should be capable of executing Structured Query Language (SQL) commands on a database.

Requirement 1: reading XML

Because we make use of the eXtensible Markup Language (XML) to transform data from one format to another, the enactment mechanism should include the possibility to read and understand XML, as well as reading an XML file. Because XML was already introduced in 1996, every respected and comprehensive programming language includes an option to reading XML files by default.

Requirement 2: understanding SQL

The enactment mechanism will transform data from the XML file to the object model of the CASE tool. When this process is completed, the data needs to be stored in the repository of the CASE tool externally. By default, Jira utilizes a central repository that contains the data that is entered in the tool itself. Modifying data through the interface of Jira is simply executing queries on the database. Since we mimic this process with the use of the enactment mechanism, the enactment mechanism should be capable of executing queries on the repository externally in the same way as Jira this accomplishes.

3.4.2 Enactment mechanism conclusion

Taking into account the aforementioned requirements, we can choose between every respected and well-known programming language. Today we can find over one hundred programming languages on the web ranging from very simple languages to highly sophisticated languages with which you can develop any kind of software.

If we take a look at Jira we see that Jira is developed entirely with the use of the programming language called Java. Although we are not concerned with this programming language in this research, it would certainly be the most optimal solution to mitigate any future problems. Unfortunately we had hardly any experience of Java and could therefore not use this programming language as our main language. Nevertheless we do have a rather profound understanding of PHP[5]. Similar to Java, PHP also supports Object Oriented Programming (OOP), though in a less strict manner in comparison with Java. Still, the object model that someone develops in PHP can easily be converted to Java when making the right changes.

By default PHP includes an XML reader which we can use for reading the XML file and eliciting the necessary data. In conjunction with MySQL we can consequently store the transformed data in the repository of Jira externally in a way that is understood by the CASE tool. Because PHP also allows us to achieve our research goal, we select this programming language as the main language for our mechanism.

3.5 Architecture for Method Enactment

In Figure 8, the general architecture for method enactment is shown. On the left side of the diagram, the CAME tool MetaEdit+ is shown. On the right side of the diagram, the CASE tool Jira is shown. Both tools have their respective repository for storing and retrieving data. In both repositories, the meta-data as well as the regular data of the respective tool is stored.

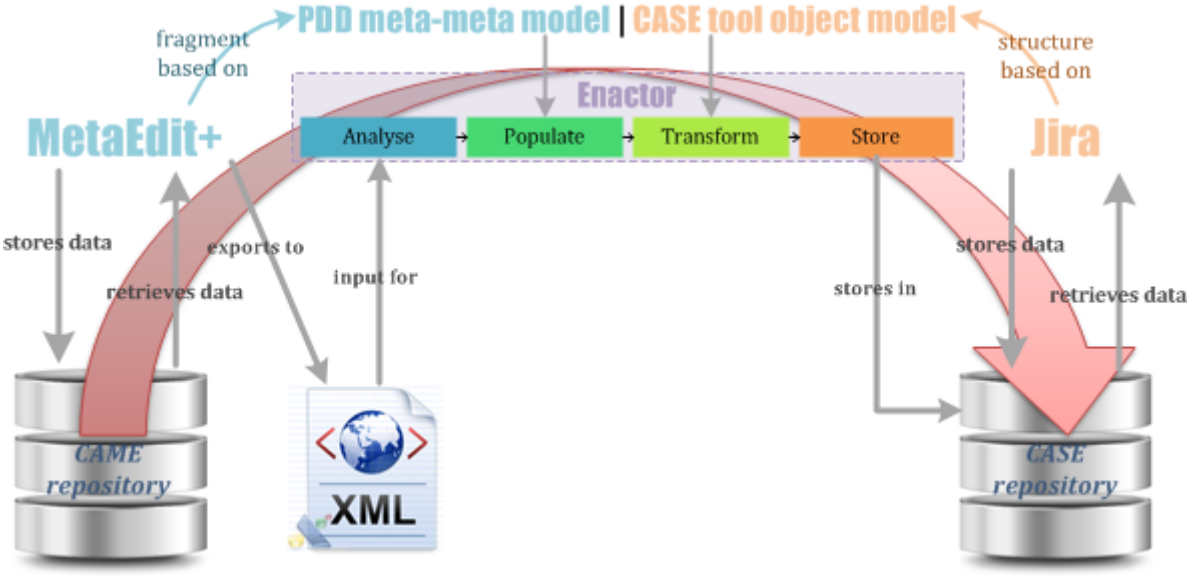


Figure 8. Architecture for method enactment.

[5] http://php.net

The enactment process goes from left to right, indicated through a red marked arrow. At first, the fragment data that is stored in the repository of the CAME tool is retrieved. Based on this fragment data, an XML file is generated and created. The XML file is based on the GOPRR structure and is temporarily stored on the computer. This XML file contains all the data associated with the fragment. The XML file is analysed by the enactment mechanism that extracts and populates all the required data for this research. During the population process, the data is temporarily stored in objects according to the PDD meta-meta model (Weerd *et al.,* 2007). When the population process is finished, the data will be transformed to data that is in accordance with the Jira object model. Based on this object model, the data is stored definitive in the repository of Jira by using MySQL. When this process is completed, Jira can retrieve the just added data from the repository.

3.6 Tool characteristics

Before we can proceed to the enactment approach, the functions and possibilities of MetaEdit+ and Jira need to be investigated in order to get a better understanding of how they can be of value for the remainder of this research. The following subsections elaborate on the functions, possibilities and characteristics of the selected tools.

3.6.1 MetaEdit+

The first version of MetaEdit+ was released in 1993 were it encompassed a very few features to domain specific modelling. Over the last two decades however, MetaEdit+ has almost become the standard tool for domain specific modelling that has been used in numerous prior researches (e.g. Oinas-Kukkonen, 1996; Tolvanen, 2004) due to its configurable method engineering environment. It enables the user to build its own modelling tools and code generators without writing a single line of code (Tolvanen & Rossi, 2003). MetaEdit+ was originally developed to overcome the software development problems that arose with regular CASE tools (such as insufficient quality and low productivity during development) and to enhance method specification, integration, management and re-use (Kelly *et al.,* 1996).

MetaEdit+ is multi-method, multi-user and multi-tool environment for computer aided software engineering and computer aided method engineering. To support the engineer in this task, MetaEdit+ includes a *workbench* to support the method engineering environment and a *modeller* to support the software engineering environment. The MetaEdit+ workbench is used for the development of your own modelling language, which is stored as a meta-model in the repository. This meta-model can be used in MetaEdit+ *modeller* for the creation of your own domain specific model. The MetaEdit+ workbench was used in the research of Vlaanderen (2010) where he created the language objects, properties, associated rules and symbols of a PDD through the GUI of MetaEdit+. All this data is stored as a meta-model in the repository of MetaEdit+. Because this research also focuses on the creation of valid PDDs, we will use this meta-model to base our fragments on.

The MetaEdit+ modeller follows the given model language that is defined in the workbench suite, i.e. its meta-model. With the use of MetaEdit+ modeller, the actual methods (in our case a PDD) can be created using MetaEdit+'s built-in diagram editors, browsers or generators. In other words, we can create a PDD by using the objects, properties, associated rules and symbols that are defined in the meta-model of a PDD. All this meta-data is retrieved from the repository of

MetaEdit+ during the development of the method. When the development of the method is complete, its respective data is stored in the same repository.

3.6.2 Jira

Jira is a proprietary bug tracking and project tracking tool for software development. It has been developed since 2002 by Atlassian and can be accessed through a regular web browser. According to the Jira Atlassian website, Jira serves more than 20.000 companies worldwide.

By default, Jira comes with an extensive amount of features that can be customized to a specific organization or person. With the use of a dashboard, each user has the possibility to customize its own visibility and with the right restrictions, each user can view, create and modify existing *issues*. An issue is a record in the repository of Jira and the name refers to an issue that can arise during the development of software. With the use of different *issue types* (e.g. a bug, feature or task), a user can add a specific concern to the repository of Jira. Also, administrators can create custom-made issue types.

In Figure 9, a graphical overview of the basic features of Jira is provided. To configure Jira, an administrator can create a custom field, for instance a field called *requirement type.* Before creating a field, the administrator can select the type of field, for instance a text field, radio buttons or a select list. Also, default value(s) can be added to a field, e.g. the options *business requirement* or *market requirement* as default values for the *requirement type* field. Fields can be placed on one or more created screen(s). For instance, you can create a screen called *requirement* and place the field *requirement type* on this screen. When completed, a screen can be used twofold. It can either be used in a *workflow* or mapped to a *screen scheme*. A *screen scheme* is used to choose which screen will be shown when a user performs an issue operation (e.g. create/view/edit issue); for instance when you click on *create issue,* you can select that the *requirement* screen needs to be shown. A screen that is placed in a workflow shows up when you enter a particular status in the workflow. Workflows exist out of *statuses*, which represent a current state in the workflow and out of *transitions,* the type of transition between different statuses. For example, if you have an activity in the

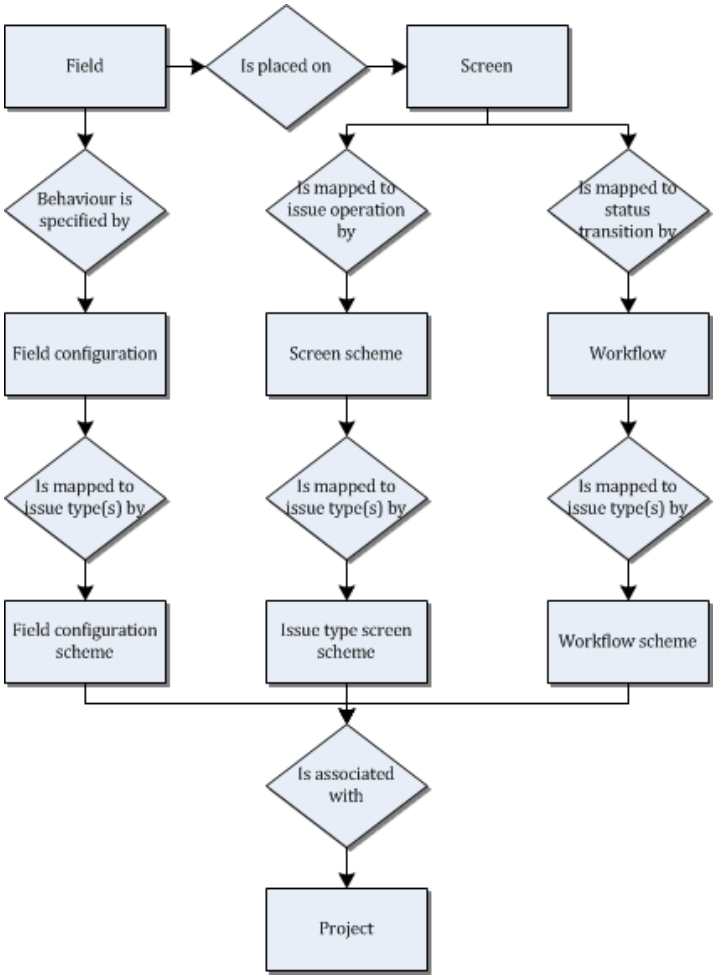


Figure 9. Overivew of Jira's features.

workflow called *create requirement,* you can associate the screen *requirement* with this activity. The *requirement* screen will then show up when you enter the *create requirement* activity.

Finally, a *workflow scheme*, *issue type screen scheme* and a *field configuration scheme* can be associated with one or more created project(s) so that each data is kept within the corresponding project. In addition to the above mentioned basic features of Jira, Jira can also be extended by importing (third party) plugins to extend Jira with more functionality of features. Also, different types of users or user groups can be created where each user or user group has its own restrictions. Finally a comprehensive amount of basic filters, settings and configuration possibilities are available next to a default import/exportation function for importing or exporting all the existing data in or from the repository of Jira.

Workflows and *screen schemes* are mapped to *workflow schemes* and *issue type screens* respectively. Both schemes are collections of multiple workflows or multiple issue type screens that can be used throughout Jira respectively. A field on a screen can be enabled or disabled using the *field configuration*. Therefore, a field that is placed on a screen can still be invisible if the field configuration is configured so. The field configuration (which stores all the visibility of the fields) is mapped to a *field configuration scheme*. The field configuration scheme maps the field configuration to issue types in a project. For instance, if you create an issue type called *requirement process,* you can add the field configuration to this issue type. In that case, all the fields that are configured on the *field configuration* will be shown to the user when you create/view/edit the associated issue type.

CHAPTER 4

Incremental Method Enactment

This section elaborates on the approach towards the enactment of a method fragment, based on the approach we sketched in Figure 7. In each corresponding section, a miniature depicts the progress in the enactment process. First, this section will elaborate on the configuration of our research environment. Next, we elaborate on our used mapping for mapping a fragment to Jira and the used method fragments within this research. Afterwards we will elaborate on the generator of MetaEdit+ that we utilized to invoke our enactment mechanism. Subsequently, the used mechanism, i.e. the enactment mechanism, will be extensively elaborated upon to show its working. Finally, we will elucidate on the results in Jira and the data implications that arise when our presented approach is employed.

4.1 Configuring the research environment

The research environment that is subject to this research is a local machine with an internet connection that runs Windows 7 Ultimate. As we are also using the programming language PHP to develop our mechanism, we need to have a parser that can read and execute our PHP code. Therefore we installed WAMPServer. WAMPserver is a windows web environment that includes Apache, PHP support and MySQL support and runs as an active service on the Windows machine. The apache environment will be used to parse the PHP code that we are creating. The MySQL environment will be used as our primary database in which the data shall be stored. With the use of the phpMyAdmin web interface that comes along with WAMPServer, we created a database called *enactment_jira* that we will use as our primary database for storing the data of Jira.

4.1.1 Configuring MetaEdit+

Besides the usage of WAMPServer, we downloaded and installed the full MetaEdit+ Workbench package from the site of MetaCase. When the installation of the software was completed, we created our own repository that we are solely using for this research. After opening our repository, we first imported all the objects belonging to the meta-model of a PDD. The objects were created in a prior research that was carried out by Vlaanderen (2010). In addition to the available objects, the rules and constraints that are subject for creating a valid PDD are also included. When importing was finished, we gained the possibility of creating valid PDDs through the Graphical User Interface (GUI) of MetaEdit+.

4.1.2 Configuring Jira

After installing MetaEdit+ and WAMPServer, we downloaded Jira version 4.4.4 from the Atlassian website. With regard to the installation of Jira, there are two different ways in which Jira can be installed: either local or dedicated. If we would choose to install Jira locally, Jira can only be accessed and used by the computer on which Jira is installed. Also you are not able to

access the repository's data (which is based on the HyperSQLDB database engine) from an external computer. As this solution is useless because Jira is typically used throughout an organization by multiple users, we want Jira to be installed in a dedicated environment with the use of an external database. In this way, a Jira installation can be used throughout an organization and used by different users.

By default Jira offers a vast amount of options for connecting it to an external database such as PostgreSQL, MySQL, Oracle or Microsoft SQL server. For Jira, it does not matter to which database it is connected as it is rather a preference of the user. As we have a good understanding of MySQL, we selected this as our database type. Also, MySQL offered us several advantages:

1. MySQL 5.0 supports the rollback of a transition. If one query fails during execution, all the previous executed queries are rolled back. In this way we avoid pollution in the database if something goes wrong anywhere in the process. This will prevent a situation where Jira would cause strange behaviour due to unfinished changes in the data.
2. MySQL can be accessed through various programming languages such as PHP, ASP, Python and Java.
3. Full control over the database and its structure through a simple web user interface.

Because we are using an external MySQL database to store all the data Jira produces, we created our own database. However, because this database is manually created and thus empty, we need to obtain the table structure of Jira first.

We obtained the table structure of Jira by first installing Jira locally. When the installation of Jira was completed we immediately exported all our Jira data to XML with the use of the default Jira exportation function. This XML data contained the table structure as well as the data that is entered in Jira. Because we did not enter any data in Jira, only the table structure was present in the XML. Next we reinstalled Jira and selected the option for using Jira in a dedicated environment. During this installation, we entered the path to our created *enactment_jira* MySQL database and selected the option to import data from a previous installation. We consequently selected our created XML and let Jira import all the data from the file. When this process was finished, we had connected Jira to our external MySQL database with its respective table structure.

4.2 Mapping and developing the method increments

When the research environment was configured, we could create the method fragments. The following sections elucidate on the mapping of fragments to the tool fragments of Jira and our used increments in this research.

4.2.1 Mapping the fragments to Jira

Before the enactment can take place, we first reviewed the adaptability and functionality of Jira. Based on this information, we created a mapping that allowed us to convert the data of a fragment to the features that are available in Jira. Figure 10 depicts the mapping.

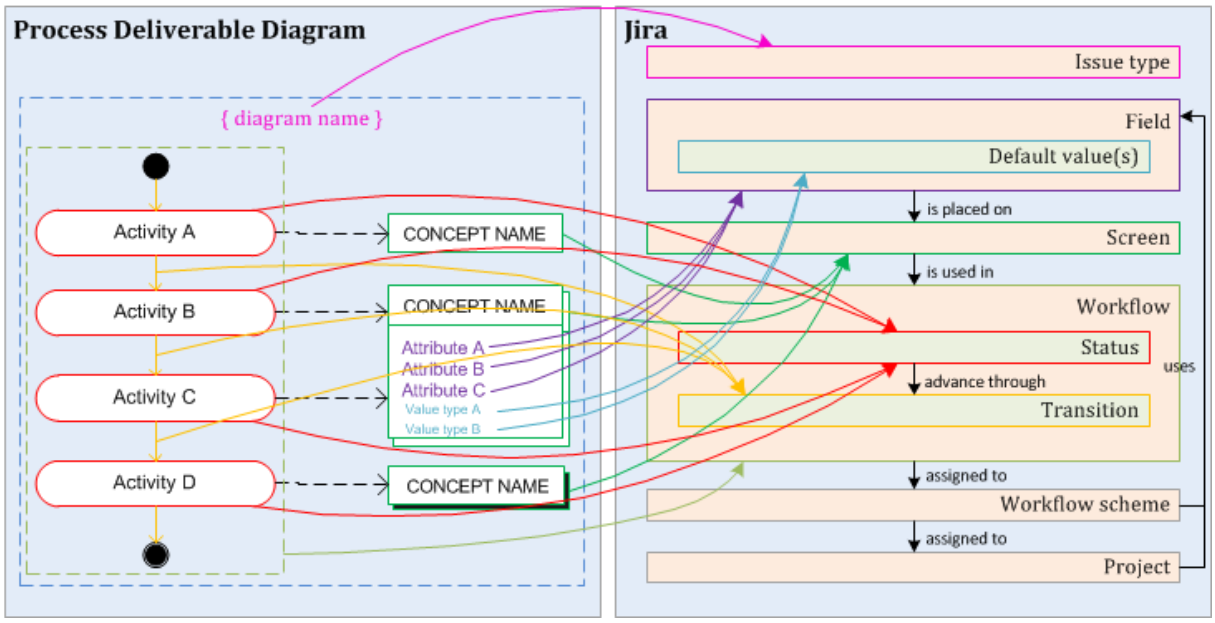


Figure 10. Mapping of the PDD fragments to Jira's fragments.

On the left-hand side of Figure 10 a simplified fragment in the form of a PDD is provided, encircled with a blue dotted line. On the right-hand side of Figure 10, the default features that are available in Jira are shown. The coloured line represents the relationship of the method fragments of a PDD in relation to the tool fragments of Jira. Table 6 provides a textual explanation of the relationship between both fragments. Per mapping, we will elaborate on the usage by providing it with an example of the fragment that is shown in Figure 11.

Table 6. Textual mapping of the fragments to Jira's fragments.

#	Fragment	Relationship	Jira tool fragment	Usage domain
1	Diagram name	is used to create an	Issue type	per project
2	Process side	will produce a	Workflow	per project
3	Activity	is used to create a	Status	in a workflow
4	Transition	is used to create a	Transition	in a workflow
5	Concept name	is used to create a	Screen	per project
6	Attribute	is used to create a	Field	per project
7	Value type	will be used as (a)	Default value(s)	per field

1: diagram name to issue type

The diagram name of the fragment that a user creates will be used to create an issue type in Jira. The usage is as follows:

Fragment data in MetaEdit+		**Issue type name in Jira (example)**
{ diagram name }	=	*Requirement management process*

For instance, if you create a first version of a fragment that is entitled '*Requirement management process*', this diagram name will be used to create an issue type in Jira. All the data that is entered in Jira underneath this issue type belongs to that specific fragment. When an updated version of

the fragment is created, the existing issue type will not be affected. Issue types can be used in different projects or in one project only.

2: process side to workflow

The complete process side of the fragment (which is marked with a light green dotted line in Figure 10) will be used to create a workflow in Jira. A workflow in Jira is only represented by a name and the entire workflow is based on an XML structure. Because the creation of the XML is rather comprehensive, we will address this part of this research in section X.X. The usage for creating a workflow in Jira is as follows:

Fragment data in MetaEdit+		**Workflow name in Jira (example)**
{ diagram name } + "workflow"	=	*Requirement management process workflow*

A workflow in Jira is similar to the process side that is modelled in a PDD. A workflow exists out of *statuses* and *transitions.* A status is a step in the workflow through which you advance while working in Jira. For instance, if you completed activity A in the workflow and marks this as complete, you advance to the next status (i.e. the next activity). Advancing through statuses in Jira requires *transitions.* Transitions are moments in the workflow in which you can complete one activity and continue with the next activity.

3: activity to status

An activity within a PDD is used to create a status in Jira. The status in Jira is a direct copy of the activity as described in the PDD, namely:

Fragment data in MetaEdit+		**Status name in Jira (example)**
{ activity name }	=	*Gather requirements*

Statuses in Jira are used within workflows. They are used to indicate at which point the user currently is in the entire workflow; similar to a PDD. Each status is linked to a screen so that a status can present information to the user.

4: transition to transition

A transition in a PDD is the same as a transition in a workflow in Jira. They are used to close an activity a user is currently dealing with and to continue with the next activity in the workflow. The creation of a transition is as follows:

Fragment data in MetaEdit+		**Transition name in Jira (example)**
{ activity name of the upcoming activity }	=	*Write draft version definition*

The name of the transition is based on the upcoming activity. For example, if you are at the *Gather requirements* in the workflow, the transition name will be *Write draft version definition* if this is the upcoming activity (see Figure 11 for an example how such a fragment).

5: concept name to screen

The name of a concept is used to create a screen in Jira. For instance, if you have a concept called REQUIREMENT, a screen is created with the name REQUIREMENT. A screen contains fields that are based on attributes of the concept. As such, the data is used as follows:

Fragment data in MetaEdit+		**Screen name in Jira (example)**
{ concept name }	=	*Requirement*

In the event a concept does not contain attributes, the screen does not have any input fields. Although this seems to make little sense, the screen is linked to a status that can contain many other functions such as attaching a document or adding comments to a particular status. In that case having a screen can still be very useful even though it contains no fields.

6: attribute to field

An attribute of a concept within a PDD is used to create a field with the same name. Therefore the usage is as follows:

Fragment data in MetaEdit+		**Field name in Jira (example)**
{ attribute name }	=	*Code*

The field that is created will be placed on the screen of the respective concept. For instance, if a concept called REQUIREMENT has an attribute called *code*, a screen is created (namely REQUIREMENT) which has a field called *code* (see Figure 11 for an example of such a fragment).

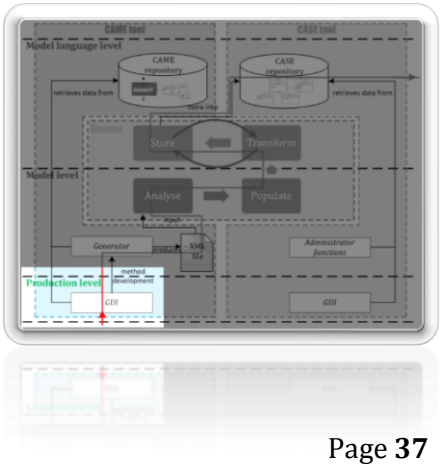
7: value type to default value(s)

If an attribute has attributes, these attributes will be used as default values of the respective field. For instance, if the attribute *code* has attributes such as G1, G2 or G3, a select box field is created that contains the values G1, G2 of G3. These values can be selected from a drop down box for instance. The data in Jira is derived as follows:

Fragment data in MetaEdit+		**Default value(s) in Jira (example)**
{ value type name }	=	*G1*

4.2.2 The method increments

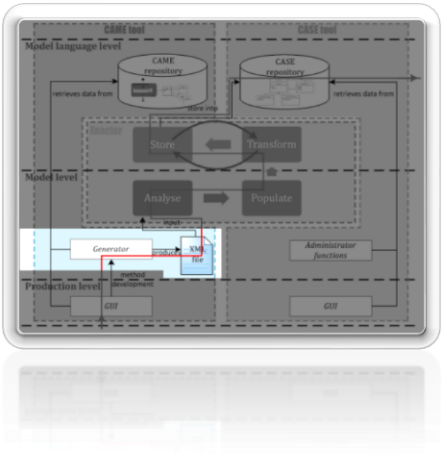
With the use of the Graphical User Interface (GUI) from MetaEdit+, we modelled five different fragments (depicted in appendix I) that we used as input for our enactment mechanism. Because this research focuses on the prioritization of requirements, we based our fragments on a requirements prioritization process as it could occur within an organization. To emphasize the notion of mimicking a real-life setting, we selected the first five fragments from a previous conducted case study at Infor Global Solution; a vendor of ERP software. This case study was conducted



through Weerd *et al.* in 2010. The fragments show an evolutionary approach of a requirement prioritization process that advances from a very simple process to a rather complicated comprehensive process. The used fragments are depicted in appendix I and a textual elaboration of each fragment can be found in Weerd *et al.* (2010).

4.3 Utilizing MetaEdit+'s generator

After the development of the method fragments, we want MetaEdit+ to generate the XML file and to invoke the enactment mechanism. To accomplish this, we utilized the generator of MetaEdit+ to create our own custom-made functionality. We created a new generator which we called *!Enact*, referring to the enactment of the current fragment you are viewing. After creating this generator, MetaEdit+ opens a new dialog containing the following code:




```
Report '!Enact'

endreport
```

We deliberately put an exclamation mark in front of the word *Enact* because using an exclamation mark has a purpose: it tells MetaEdit+ that we want this function to be visible on the MetaEdit+ toolbar through a button. Between the *Report* and *endreport* space, the desired commands can be written down that are read in a chronological order. As we mentioned before, we want to export our diagram to XML first. Hence we added the following command to our function:

```
filename 'c:\wamp\www\pdddata\' id '.mxm' print
```

This command tells MetaEdit+ that we want to write the current diagram to a *.mxm* document and save it to our predefined folder. The *.mxm* extension is the default XML export extension of MetaEdit+. Although the filename has no *.xml* extension, the data within the file is certainly based on the notion of XML. The usage of the command *id* refers to the current name of the diagram. In summary, the above command exports the current diagram to an XML document, places it in the *pdddata* folder and the file has the same name as the name of the diagram. In addition to this command, we also added a second command to the generator:

```
filename 'c:\wamp\www\pdddata\' id '.png' print
```

This command has the same meaning as the previous command; however it writes the current diagram to an image (with a *.png* extension) instead of an XML document. Since we use the generated image for visualization purposes, this step can be omitted if desired. It is not a mandatory prerequisite for the functionality within this research.

Finally, we added a third command to our generator:

```
external 'firefox.exe "localhost/?file=' id '.mxm"' execute
```

This command tells MetaEdit+ that it should execute an external program (which is in this case Firefox) and opens the location next to it. In this case, Firefox opens the index file that is present on the *localhost* server along with a parameter (called *file*) that refers to the just generated XML document. In total, our generator has the following code:

```
Report '!Enact'
      filename 'c:\wamp\www\pdddata\' id '.mxm' print
      filename 'c:\wamp\www\pdddata\' id '.png' print
      external 'firefox.exe "localhost/?file=' id '.mxm"' execute
endreport
```

When we save this generator, a button is created on the MetaEdit+ toolbar (see Figure 11; indicated with red border).

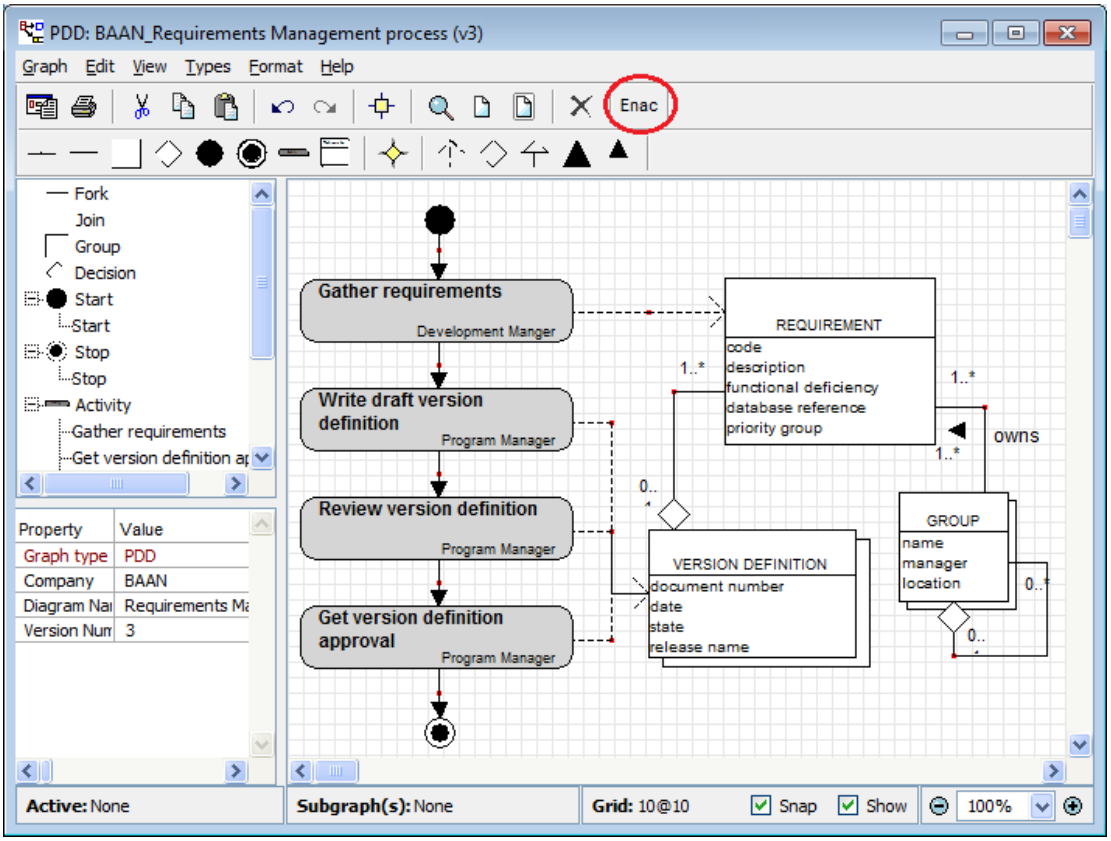


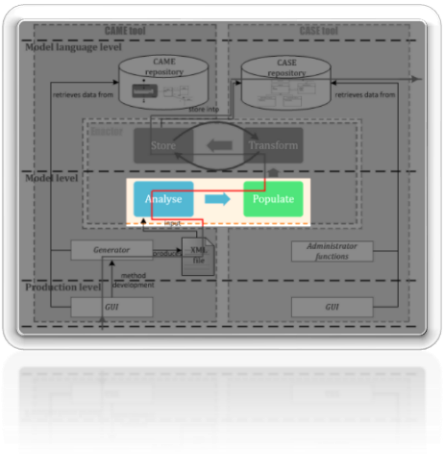
Figure 11. MetaEdit+'s GUI showing a fragment and the enactment button.

Pressing the *Enac* button in MetaEdit+ will start the generator and execute the commands that we wrote according the notion of MERL. In this research, it will generate the XML file first with an .MXM extension, consequently generate a PNG file of the fragment and finally start Firefox. The l*ocalhost* location in the URL refers to our apache installation of WAMPServer, similar to any other URL on the internet.

4.4 The enactment mechanism

When the button is pressed in MetaEdit+, the enactment mechanism is triggered. The subsequent sections elaborate on our developed mechanism by explaining each of the four phases that we visualized in Figure 7, namely *analyse, populate, transform* and *store.*

4.4.1 Analysing and populating the XML file



When a fragment is created in MetaEdit+ (see Figure 11 for an example) and the user presses the *Enac* button, MetaEdit+ will process the commands that we wrote in MERL and open Firefox. When Firefox is opened, the respective URL that was sent from MetaEdit+ (which refers to the XML file) is loaded.

The index file of the URL invokes an *Extractor* class (see appendix II for an overview of the used classes in the form of a UML class diagram). The Extractor class creates an instance of the class *Xml.* After instantiating the Xml class, this class will create an instance of the *SimpleXMLElement* class (which is available in PHP by default) with which a developer can iterate through the XML file with the use of various functions. Via this instance, the Xml class will elicit (1) the company who made the fragment, (2) the name of the fragment and (3) the version of the fragment. These variables are used in the Extractor to create a *Method* object (note that we are using the word *object* to indicate an instance of a class because the development design is based on the Object Oriented Programming (OOP) principle). The Method object contains all the data of a PDD in different objects and the structure is based on the PDD meta-meta model as created by Weerd *et al.* (2007). However, because we are also using attributes that can contain attributes (i.e. value types, see Figure 10 for details), we extended the meta-meta model of Weerd *et al.,* (2007) with a *SubProperty* class as a generalization of the *DeliverableFragment* class. Therefore, a total of 16 classes are needed to store the data of a fragment in its respective objects. With the use of five iterative functions in the Extractor class (see appendix II), the class extracts all the necessary method fragments of a fragment and stores them in the respective instances of a class. Table 7 provides an overview of which method fragment of a PDD uses which class.

Table 7. Data transformation table.

PDD method fragment	Class
Start, stop, decision	ControlNode
Activity	Activity
Role	Role
Concept	Concept
Property / attribute	Property
Value type	SubProperty
Control flow	Transition
Relationship	Relationship

As we can see in Table 7, only data that is useful in a fragment will be processed. Therefore only eight method fragments of a fragment are subject to extraction. A start, stop and decision are

stored as a control node as these nodes are controlling the flow of a process. Furthermore we distinguish between a *control flow* and a r*elationship.* A control flow is the flow of a process. It indicates which activity/control node is connected to which other activity/control node. A relationship is the dotted arrow between an activity and a concept. It tells which activity produces which concept.

When the extraction process is finished, each object (referred to as an instance of a class) contains its respective data. However, because we also want this data to be accessible throughout our script, we temporarily store all this data in a session. In this way, we can access our data from anywhere in the script at any point in time. To get a more profound understanding of the approach used for extracting the data from the XML file, appendix II provides a comprehensive overview of the used classes, its attributes and its respective functions in the form of a UML class diagram (Object Management Group, 2004).

4.4.2 The enactment mechanism GUI

When the extraction process is finished, all the data is presented to the user with the use of the HyperText Markup Language (HTML) in conjunction with jQuery in such a way that it is understandable by a human. Figure 12 presents the GUI of the enactment mechanism.

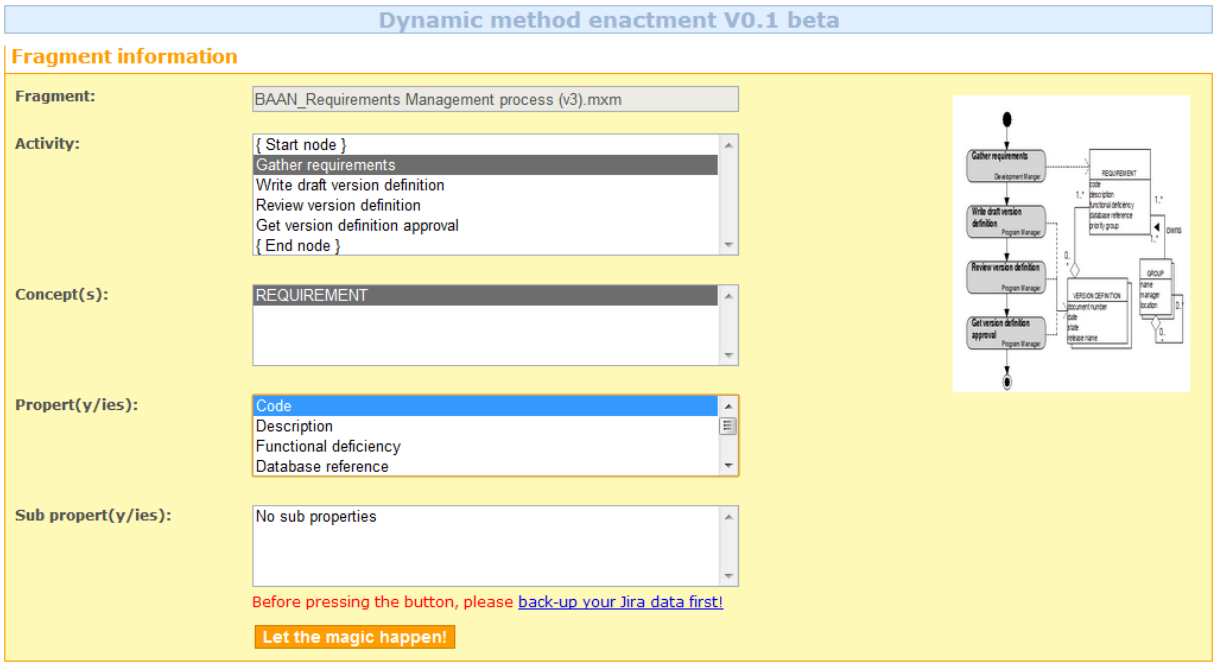


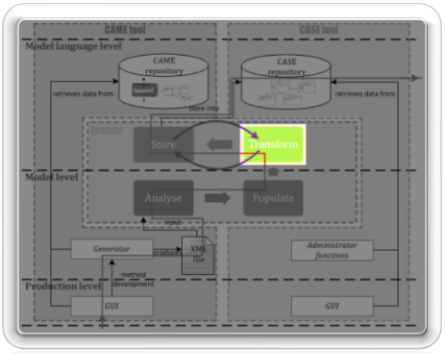
Figure 12. The GUI of the enactment mechanism.

The GUI presents the user with the desired information in a section called *Fragment information*. In this section, the current increment that was invoked through MetaEdit+ is shown as a graphical image on the right of the screen and can be enlarged by clicking on it. Along with the filename of the fragment, the image is only present to help the product manager avoid a situation where he or she would accidentally open the wrong fragment from MetaEdit+. Additionally, all the activities that are present in the PDD are visualized in *activity* select box and have the same order as they occur in the PDD. As the visualization is based on a valid PDD, each fragment a user creates should comply with the modelling standards of a PDD. Therefore it

should contain at least a start node, one activity and an end node. Between both nodes, the activities are shown.

The user can click on each activity in order to show its related concepts. For instance, the *'Gather requirements'* activity has a concept called REQUIREMENT. If the user clicks on this activity, the REQUIREMENT concept will be showed in the concepts section. When the user consequently clicks on the REQUIREMENT concept, the properties of that particular concept are showed in the *properties* field. If a property contains value types, the user can also click on this property. If there are no value types available, a notification in that particular section will provide the user with this information. In this way of navigating through a PDD, a user can completely walk through all the corresponding activities, concepts, attributes and value types in a textual manner with the use of the GUI. This way of navigating aids the user in checking whether all the information that was entered in the PDD is presented in a correct manner.

4.4.3 Transforming data from the PDD meta-meta model to the Jira object model



When all the presented information that appears on the screen is correct, the user can click on the button to start the enactment process. However, before the button is pressed, it is recommended to back-up all the data in Jira first. In case something fails during the operation, the made back-up can always be restored. In that case you avoid a situation where the existing data would be contaminated or gone. Additionally, it is recommended to stop the Jira service from running on the computer to make sure that the data within the repository of Jira is accessible externally.

When the button is pressed, an *Enactor* class is invoked. The Enactor class triggers the entire enactment process. The concept behind the enactment process is as follows. First, we determined which tables in the repository of Jira required data in order to achieve our research goal. In order to determine the right tables, we mimicked the entire enactment process manually and afterwards, we consulted the log file that Jira creates by default. In the log file, all the queries that are sent to the repository of Jira are written down. Based on this information, we concluded that 17 out of 118 tables of Jira required data to achieve the same result. The tables are also listed in appendix III.

For each table, we made three classes that we based on the Model-View-Controller (MVC) design pattern, namely a Value Object (VO) class, a Data Access Object (DAO) class and a Controller class. The VO class contains the actual data of a row in the table of Jira with their respective *get* and *set* functions for getting and setting data in an instantiation of the respective class. The class per table is a direct copy of the Jira table, with the addition of the VO in the class name. Figure 13 depicts this routing by showing an example with the Fieldscreen table.

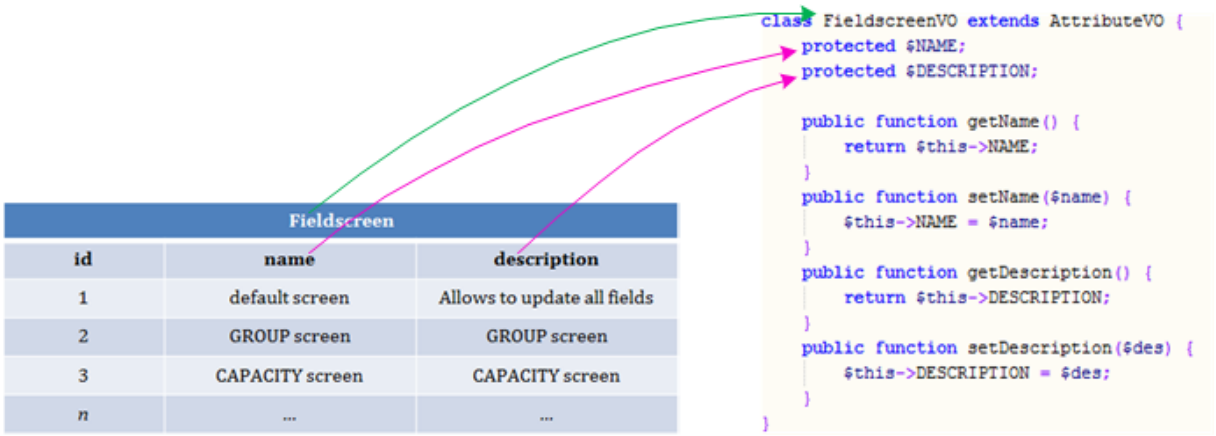


Figure 13. Mapping the Jira Fieldscreen table to the FieldscreenVO class.

The table name of Jira is used as the class name with the addition of VO and the columns of the table are properties of that class. Note that in Figure 13 the column *id* is not linked to the *id* property from its respective class as this property is inherited from its parent: the *AttributeVO* class (see also appendix III). Each row in the Fieldscreen table is a new instantiation of the FieldscreenVO class. For instance, when there are 10 field screens in Jira, there are 10 records in the fieldscreen table of Jira's repository and this will result in ten instantiations of the FieldscreenVO class where each instantiation contains the values of the respective row.

The DAO class is a class that only communicates with the repository of Jira. It only stores and retrieves data from the repository of Jira with the use of a save() function. The Controller class is responsible for controlling the data, e.g. determining whether data already exists or not and whether it needs to store data in the repository of Jira or not. For instance, for the *Fieldscreen* table (which stores all the screens that are present in Jira), we created a FieldscreenVO class, a FieldscreenDAO class and a FieldscreenController class. The FieldscreenController class is the actual controller of the fieldscreen data in Jira's repository and this class controls all the operations of the fieldscreen table. The FieldscreenVO class and FieldscreenDAO class are only in place to serve the necessary data to the FieldscreenController.

In total, 51 classes are in place, namely 17 tables multiplied by a VO, DAO and Controller class. A list of the used classes can also be found in appendix III.

In Table 8, we provided an overview of the used classes and functions for the Fieldscreen table. For the other 16 tables, the same classes and functions are created as well. Because the other classes have a similar structure, we do not elaborate on each class. Note that we are using the term objects regularly in this table because the programming design is based on the Object Oriented Programming (OOP) principle.

Table 8. Overview of the classes and functions per Jira table.

Class	Function	Explanation
FieldscreenVO	getName()	Gets the name of the *name* attribute of the FieldscreenVO object.
	setName($name)	Sets the name of the *name* attribute of the FieldscreenVO object.
	getDescription()	Gets the description of the *description* attribute of the FieldscreenVO object.
	setDescription($desc)	Sets the description of the *description* attribute

Class	Function	Explanation
		of the FieldscreenVO object.
FieldscreenDAO	*constructor()*	Instantiates the database connection.
	save($FieldscreenVO)	Saves a FieldscreenVO object to the database.
	getFieldscreens()	Gets all the fieldscreen records from the database and returns FieldscreenVO objects.
FieldscreenController	*constructor()*	Instantiates the FieldscreenDAO class.
	control($existData, $newData)	Determines whether the sent data already exists, whether it needs to create a new object and/or it needs to save an object.
	_create($newData)	Creates a new FieldscreenVO object based on the sent data.
	_exist($existData)	Determines whether a field screen already exists in Jira.
	_getLast()	Gets the last field screen from the Jira repository in a FieldscreenVO object.

The Enactor class creates a new instantiation of the FieldscreenController class. By invoking the control() function of the FieldscreenController class, the class checks whether the sent data already exists in the database through the use of the _exist() function. If this is the case, we do not add new data to the database. If this is not the case, it creates a new instantiation of the FieldscreenVO class by using the _create() function. When a new FieldscreenVO is created, the save() function of the FieldscreenDAO class is invoked. This function saves the just created object to the repository of Jira. The other functions in the respective classes are in place to support the operations in getting the right data at the right moment.

Note that, in this research, we do not have a delete function in place that deletes data from the repository of Jira when this data has been removed in a new fragment. Instead we hide this data so that it is not visible to the user by changing a parameter in the database. This option is available in Jira by default. We do not want to destroy existing data because destroying data would mean that would not be able to restore it. Because this process cannot be undone, we instead hide this data so that we can always display this data when we need it in a later stage.

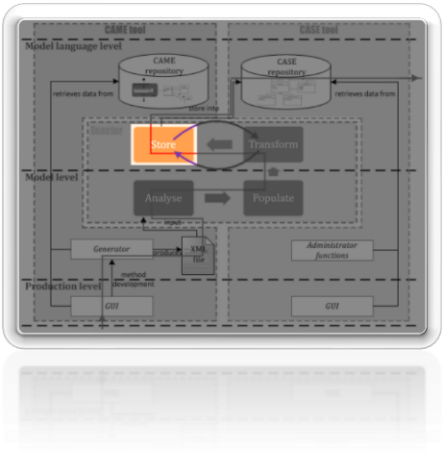
Finally, the Enactor class is responsible for instantiating all the seventeen Controller classes with the use of the *constructor.* After the seventeen instantiations, it invokes from every instantiation the control() function in the Controller class. While doing this, the Controller class handles all the operations for checking, comparing and saving the data to the repository of Jira. However, one table cannot be update immediately because an additional step is needed. The *Jiraworkflows* table stores the entire workflow of a fragment as a single XML string in the repository of Jira. Therefore, we need to reproduce the XML string by ourselves. For that reason, a *Workflowcreator* class is also in place (see appendix III). This class is responsible for creating the XML string that Jira uses as its workflow. Figure 14 shows an excerpt of how the XML string looks like that Jira uses as a workflow.

```
<step id="2" name="Gather requirements">
  <meta name="jira.status.id">8</meta>
  <actions>
    <action id="21" name="Write draft version definition" view="fieldscreen">
      <meta name="jira.description"></meta>
      <meta name="jira.fieldscreen.id">7</meta>
      <results>
        <unconditional-result old-status="Not Done" status="Done" step="3">
          <post-functions>
            <function type="class">
              <arg name="class.name">com.atlassian.jira.workflow.function.event.FireIssueEventFunction</arg>
              <arg name="eventTypeId">13</arg>
            </function>
          </post-functions>
        </unconditional-result>
      </results>
    </action>
  </actions>
</step>
```

Figure 14. An excerpt of a workflow in Jira according to the XML file format.

Figure 14 depicts the first activity (named as *step)* of the fragment that is shown in Figure 11. The first activity is called *'Gather requirements'* and it has *status id* 8 where *status id* refers to the screen in Jira called *requirement*. When this activity is finished, we see that the next activity is *'Write draft version definition'* and that activity is associated with field screen 7. Logically field screen 7 refers to the screen called '*version definition*' (see Figure 11 for this fragment). The entire XML string is produced with the Workflowcreator class and encompasses the whole process side of a PDD in a textual manner. When the XML string is generated, the enactment process continuous in the same way we described before. It is only an additional step that needs to be executed, namely the generation of the XML string (see appendix III for this class diagram). Because the process of comparing two XML strings to each other can be risky, we do not compare two entire XML strings to each other but instead replace the old XML workflow string with the new XML workflow string to make sure we have the latest XML string. For a more detailed view of the functions and classes, appendix III can be consulted.

4.4.4 Storing the data in the repository of Jira



When the enactment mechanism has created the objects according to the object model of Jira and has determined which objects need to be inserted or altered in the database, the queries can be generated and executed by using the save() function in the respective DAO class.

To insert the data from the objects into the database, we fill our queries with data from the instantiation of the respective VO class. When the query is generated, it is executed whereby it stores all the data from the VO object to the respective table. Each row in the database maintains its own unique id. When every save() function from the seventeen DAO classes are invoked from the respective Controller class, the old data in the database of Jira has been replaced with the newly created data through the enactment mechanism.

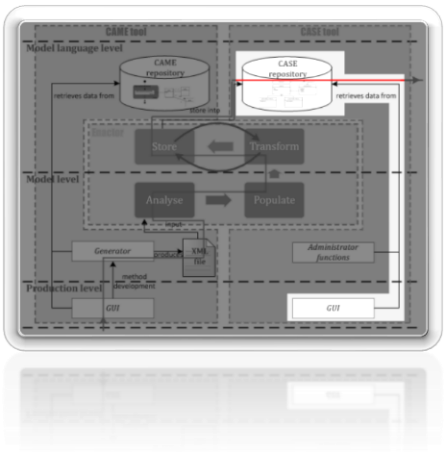
The connection to the MySQL database is handled by a *Database* class (see appendix III) that is based on the Singleton design pattern. The Singleton pattern is an approach where the class contains a static attribute that holds the database connection. The database connection is based on the Php Data Objects (PDO) extension that is available in PHP by default. In addition we used prepared SQL statements to increase safety and avoid SQL injection (i.e. attacking the database deliberately through query manipulation) by a third party.

Employing the Singleton design pattern means that the class can be instantiated once and every time the class is re-instantiated you will receive the same database connection. This allows for accessing the same database connection throughout the script without creating multiple database connections. The major advantage of this approach in conjunction with MySQL 5.0 is that we can abort the entire querying execution process when a single query fails. For instance, when we want to enact a fragment, we can start a transaction at the beginning of the querying execution process. If all the queries were executed successfully, we can commit the transaction. When a transaction is committed, all the data is definitive written to the database. In the case a query fails at any point in the query execution process, the whole query execution process can be rolled back without leaving any data behind. In that case not a single change is made to the data in the database of Jira. By employing this approach we avoid a situation where Jira would cause unexpected behaviour or other problems due to a failed enactment process.

When the enactment process has finished successfully, the Jira service can be started for the changes to take effect.

4.5 Enactment results in Jira

When the Jira service is restarted, the created and stored data through the enactment mechanism is loaded. By logging in as administrator, the created workflow, screens and fields that are created through the enactment mechanism can be consulted. When there are no current projects in Jira, the user can create a new project and associate the created workflow with this project. However, if this was already done before, you do not need to re-associate the workflow to the project.



4.5.1 Creating an issue

When a user creates a new *issue type* (issue type is the default Jira name), Figure 15 shows up.

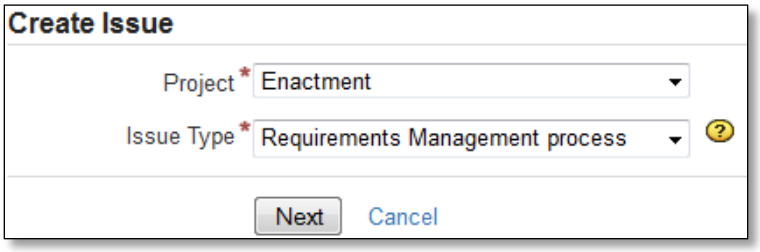


Figure 15. Creating a new Jira issue.

The project that can be selected in the first select box refers to the project that a user created beforehand. In the *Issue Type* select box, the workflow that is associated with the fragment can be selected. In this case, the name of diagram was *Requirements Management* (see Figure 11 that shows the used fragment and section 4.4.2 for the mapping of a fragment to the Jira tool fragments).

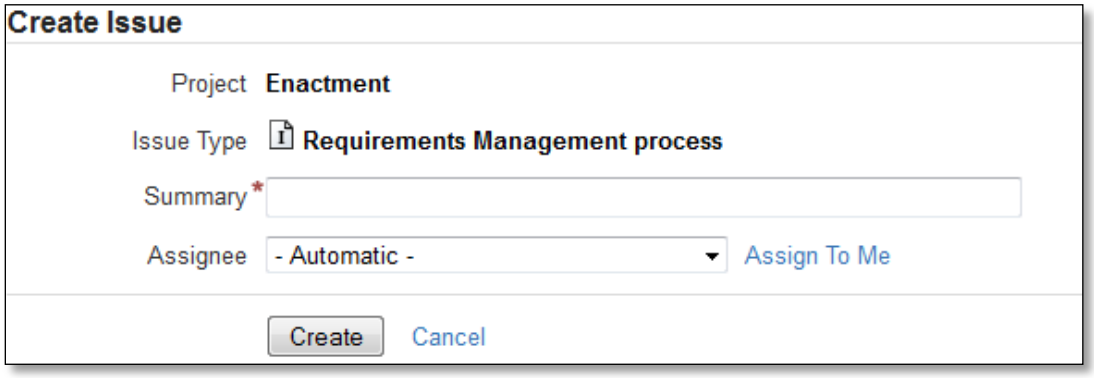


Figure 16. Creating a new Jira issue (continued).

After clicking the *Next* button, additional information (see Figure 16) can be entered such as a summary (the default name in Jira for recognition of a record) and the user that is assigned to this fragment. When an issue type is created, a generic overview of the issue type is provided to the user. Figure 17 present the general overview screen in Jira when an issue is created.

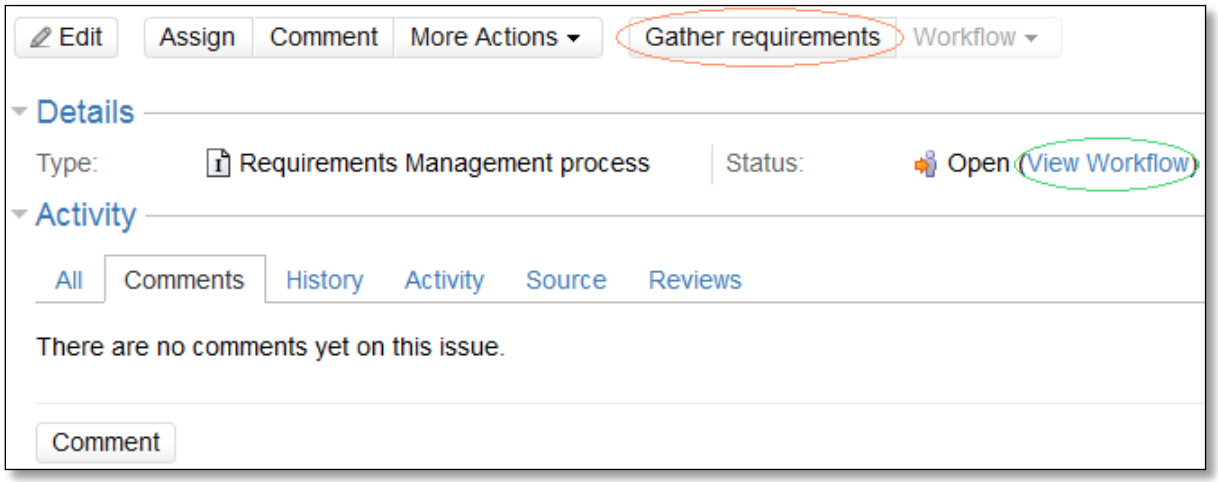


Figure 17. General overview in Jira after creating an issue.

4.5.2 Viewing an issue

In the general overview of an issue, the user has some basic functionality that is in Jira by default. By clicking on the '*View Workflow*' link (encircled in green in Figure 18), the user can determines its current status in the workflow (marked with a yellow background in the workflow on the right-hand side image of Figure 18). Also, the entire workflow is shown, starting with the '*Create*' and '*Open*' step. The creation of an issue (see previous section) is the first step in the workflow. When an issue is created, the issue advances to the '*Open*' step,

referring to the fact that the issue is open; similar to the *open* activity node in a PDD. After these two steps, the workflow is shown that is in accordance with the process side that was present in the fragment (Figure 11 shows the used fragment).

By clicking on the *'Gather requirements'* button in the general overview of the issue (encircled in orange in Figure 18), the user can advance through the associated workflow. Note that the name of this button changes throughout the workflow. In our example, the user will advance to the '*Gather requirements'* status because this is the next step in the workflow. However, before you can mark your current activity as complete, you need to enter the data that is associated with the current activity.

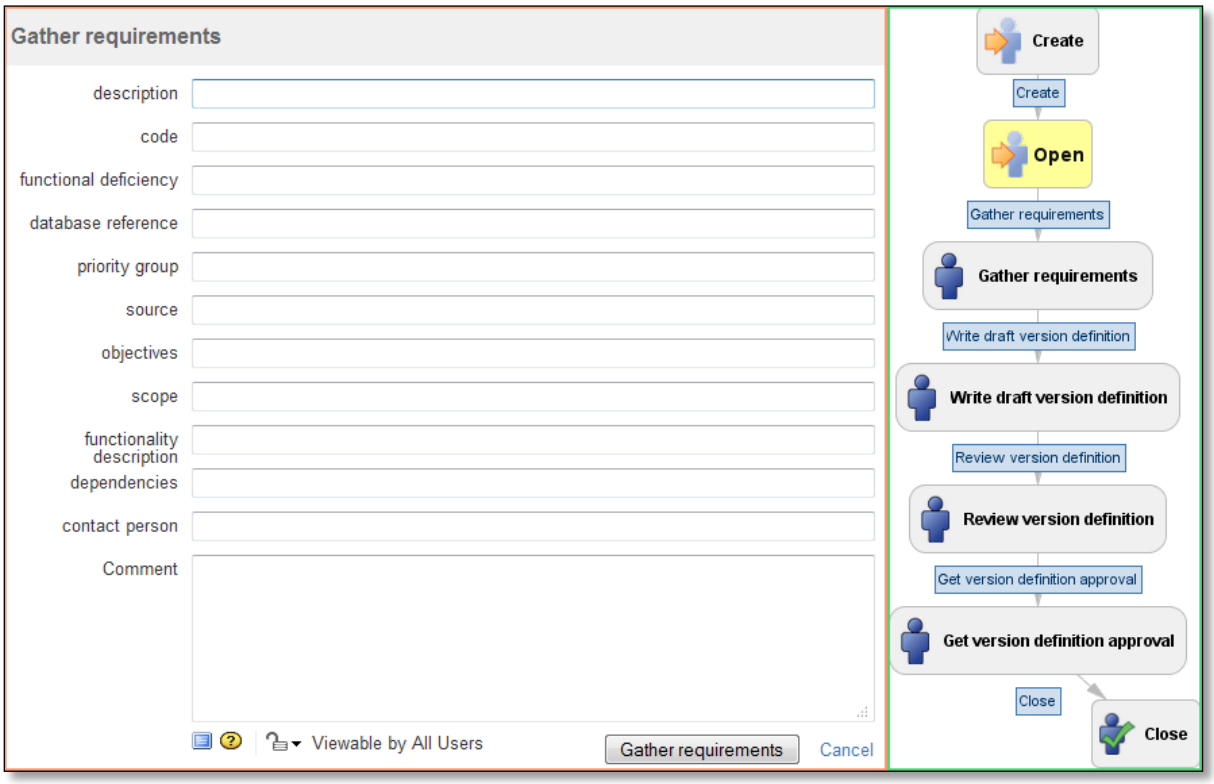


Figure 18. Lay-out of Jira after the enactment of fragment 3.

The fields that are shown to the user are based on the concept that was attached to the corresponding activity in the fragment. The left image in Figure 18 shows the fields where the user can enter its data. When the data is entered and the *'Gather requirements'* button is pressed, the user advances to the next activity. In this case, the next activity is '*Write draft version definition'*. When this activity is entered, the user can fill in all the fields that are associated with the activity. In this case, the '*version definition'* screen is presented to the user because this concept is related to the *'Write draft version definition'* step. This process can be repeated until the entire workflow is finished.

4.5.3 Incremental method evolution in Jira

When a process evolves over time, a new version of the fragment is produced. This new fragment can be enacted in the same manner as we described in the previous sections. In Figure

19, a similar overview as we provided in Figure 18 is drawn. In this case however, fragment 4 (see appendix I for the used fragments in this research) is enacted in Jira instead of fragment 3.

When the general overview is consulted in Jira after the enactment of fragment 4, no changes can be found. This is caused by the fact that fragment 3 and fragment 4 both have the same starting activity in the workflow. Therefore, no visible changes can yet be seen. When the workflow is consulted by clicking on *'View workflow'* however (see Figure 17 for this illustration), the workflow of fragment 4 is now visible instead of the workflow of fragment 3 (see Figure 19). It has replaced the workflow of fragment 3. With the *'Gather requirements'* button in the general overview, you can advance to the next step in the workflow; similar to fragment 3. Again, the current status in the workflow is marked with a yellow background. In Figure 19, it is visible that the workflow is extended with a *'Create conceptual solution'* activity; in accordance with fragment 4 (marked with a grey overlay). Also, some fields are deleted and/or added in the *'Gather requirements'* screen, e.g. the *BR number* field in Figure 19 is added (see grey overlay in Figure 19) and the *code* field is deleted. By continuously clicking on the button in the general overview and entering data in the corresponding fields, the entire workflow can be completed; similar as any other fragment in Jira.

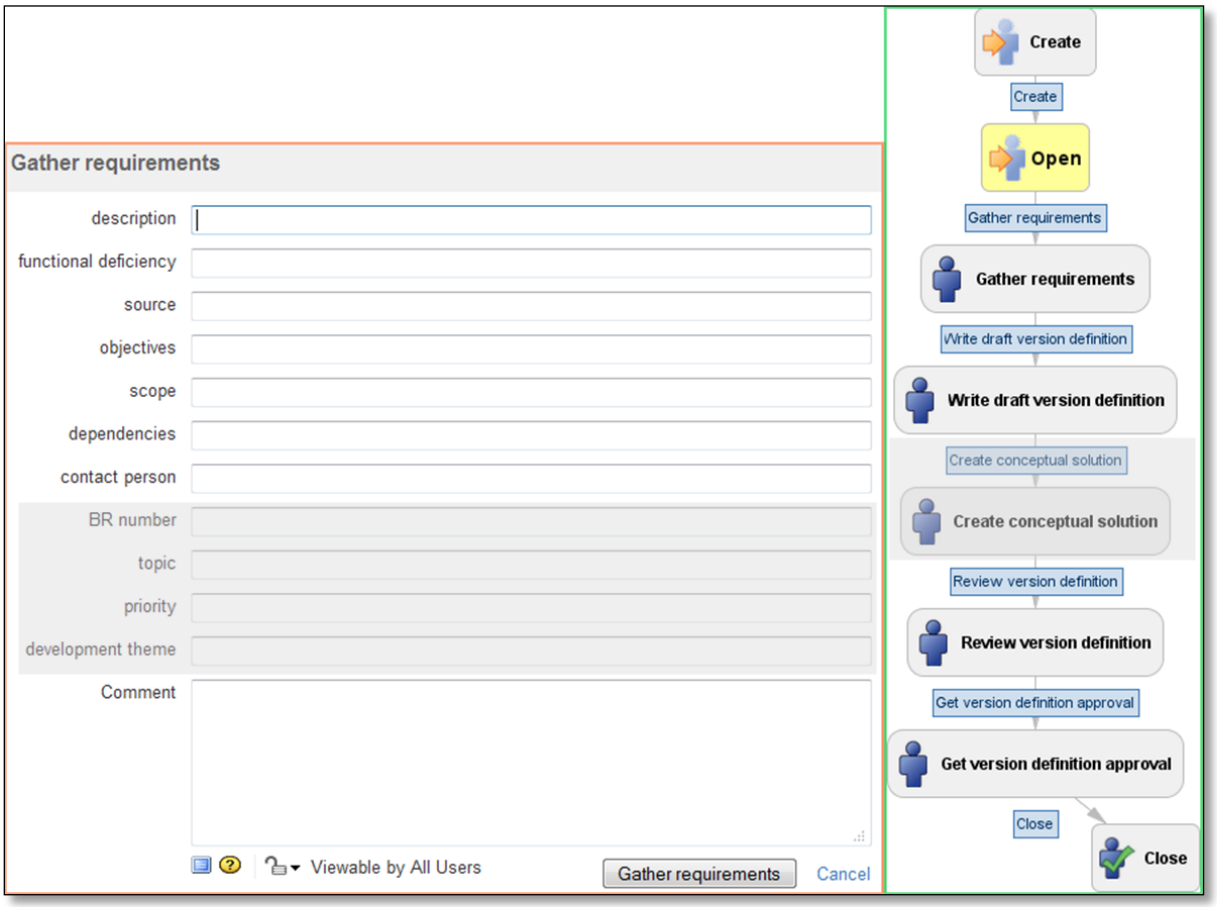


Figure 19. Lay-out of Jira after the enactment of fragment 4.

4.6 Implications on data after method enactment

When a newer fragment is successfully enacted, the data in the repository of Jira has been changed. Because the likelihood exists that the repository contained already data, this data may have been affected through the newer fragment. In our research, we replaced all the existing data within Jira with data that corresponds to the new fragment. For instance, if the lay-out of Jira is tailored to fragment 3 and we enact fragment 4 on top of fragment 3, the lay-out of Jira is tailored to fragment 4. However, this may lead to data complications. For example, if we have a screen in place called *requirement* and that screen has a field called *name*, the likelihood exists that Jira users entered data in this field, for example a user could have entered '*add exit button to the right*' as the name of a requirement. If we would delete the screen called *requirement* or delete the field called *name* because they are omitted in a next fragment, then what happens with the entered data? This section elaborates on these implications.

What we also mentioned in section 3.2.4, a fragment can evolve using 18 elementary increment types. As these elementary types cover every evolution of an increment, we base our data implications on these increment types. However, in our research we disregard the *role* increment type as we are not using this increment type in the enactment process. Instead we make use of a *value type* increment type. For that reason, the role increment type has been left out and the *value type* has been added.

Do also note that it is important to distinguish between data in Jira that belongs to the process side of a PDD and data in Jira that belongs to the deliverable side of a PDD. If data in Jira is altered that belongs to the process side of a fragment, this has no implications on the data in Jira that belongs to the deliverable side of a fragment. This is because the entire process side of a PDD is stored as a single XML string in the database of Jira. The same applies for the other way around.

Table 9 shows the data implications on existing data in the repository of Jira when a newer fragment contains newer method fragments.

Table 9. Data implications after inserting method fragments in a newer fragment.

Fragment type	Result
Concept	The insertion of a *concept* does not lead to any data complications because this only results in adding a new row to *fieldscreen* table. As a result, a new screen is created in Jira that is available to the user.
Property	The insertion of a *property* does not lead to any data complications because this only results in adding a new row to *customfield* table. As a result, a new field is created in Jira that can be associated to a screen.
Activity node	The insertion of an *activity node* will not lead to any data complications because it only affects the XML string that is used as a workflow in Jira. Therefore only the XML string is updated with additional data. Do also note that this research is constrained by using a *start* and *stop* node only because using decisions, forks and joins in a Jira workflow is a very complex undertaking to implement.
Relationship or Transition	The insertion of a *relationship* or *transition* will not lead to any data complications because the XML string that is used as a workflow in Jira is only extended with additional data. This means that only the way a workflow is carried out changes and not the data that is associated to a workflow.

Fragment type	Result
Value type	The insertion of a *value type* does not lead to any data complications because this only results in adding a new row to the *customfieldoption* table. As a result, a new default value is added to the custom value(s) of the respective field.

If an existing method fragment in a fragment is altered and this fragment is being enacted, the enactment mechanism cannot exactly determine what has changed in the new fragment. This is mainly caused by the fact that we do not have all the data of the previous fragment available to compare both fragments. As such the mechanism can only base it information on the data that is already in Jira. Table 10 shows the implications on existing data when the method fragments of a newer fragment have been updated and/or altered.

Table 10. Data implications after modifying method fragments in a newer fragment.

Fragment type	Result
Concept	The modification of a *concept* will not lead to any data complications because modifying a concept is only changing the screen name in the database of Jira. However, this only applies if the database of the CASE tool is relationally structured and makes use of unique *id*'s. Also, the fields that are associated with the respective screen may be misplaced since the screen name changed from name.
Property	The modification of a *property* will not lead to any data complications because modifying a property is only changing the fieldname in Jira. However, this also only applies if the database of the CASE tool is relationally structured and makes use of unique *id*'s. Note that in our research we create a new field in Jira and hide the old field by enabling the hide option. As such it is not visible to the user(s). We cannot delete the old field from Jira as in that case, all the data that was entered in this field cannot be shown to the user anymore because the field has disappeared.
Activity node	The modification of an *activity node* will lead to data complications on the data related to process side of a PDD in Jira because the entire process side is stored as a single XML string. Therefore the entire XML workflow string needs to be reconstructed. In this research however we are constrained by using a *start* and *stop* node only and both activities nodes cannot be modified as they are mandatory for each PDD. Still, modifying an activity node only affects the XML workflow string in Jira and not the data in the database of Jira that is related to the deliverable side of a PDD.
Relationship or Transition	The modification of a *relationship* or *transition* will lead to data complications on the data related to process side of a PDD in Jira because the entire process side is stored as a single XML string. Therefore the entire XML workflow string in Jira needs to be reconstructed. However, this also only affects the XML workflow string in Jira and not the data in the database of Jira that is related to the deliverable side of a PDD.
Value type	The modification of a *value type* can lead to data complications because the default value(s) of the respective field are changed. Therefore, data that was entered in a field with a particular value does not match anymore with the available value(s). As a result, a mismatch occurs with the existing data and available options. Note however that the entered data through the user(s) is not affected.

If an existing method fragment is deleted in a newer fragment, the enactment mechanism is able to delete these fragments from the Jira repository. Table 11 lists the implications on the data in the repository of Jira when method fragments have been deleted in a newer fragment.

Table 11. Data implications after deleting method fragments in a newer fragment.

Fragment type	Result
Concept	The deletion of a *concept* causes the screen to disappear in Jira but this does not lead to any data complications because the fields that are associated with the screen are still present in Jira. Since a screen is just a bunch of fields together, deleting a concept from the fragment does not lead to any problems besides the disappearance from the screen in Jira.
Property	The deletion of a *property* will result in the deletion of a field on the associated screen. As a result, the respective field and the data that was entered by users in that particular field are not visibly anymore. Because we do not want loose this data, we disable the visibility of the field in Jira instead of deleting it. In this way, the field and its associated data can always be restored in a later fragment. If we would delete it, all the associated data with that field would become useless.
Activity node	The deletion of an *activity node* will lead to data complications on the data related to process side of a PDD in Jira because the entire process side is stored as a single XML string. Therefore the entire XML workflow string needs to be reconstructed. For existing issues, this means that its current state is affected because its current state has been omitted from the workflow. As a result, the data is not visible anymore to user, but still present in Jira. Moreover, this existing data cannot be altered anymore.
Relationship or Transition	The deletion of a *relationship* or *transition* will lead to data complications on the data related to process side of a PDD in Jira because the entire process side is stored as a single XML string. Therefore the entire XML workflow string needs to be reconstructed. For existing issues, this means that its current state is affected because its current state is deleted from the workflow. As a result, the data is not visible anymore to user, but still present in Jira. Moreover, this existing data cannot be altered anymore.
Value type	The deletion of a *value type* results in the deletion of existing default value(s) of the respective field. As a result, some default value(s) are not available anymore to the user. Existing values in the database are not affected, but the associated default value does not match anymore with the new existing value(s). Therefore, it may occur that a different value is associated with the field than was originally selected. Also, it also may occur that the field is left blank because the field value does not exist anymore.

In the previous tables, we showed the implications of adding, modifying or deleting method fragments from a fragment. Also, it is important to mention that when a workflow changed in a newer fragment, the current state in the workflow is based on the workflow in the newer fragment. For example, if you are at the *'Write draft version definition'* step in the workflow (see Figure 11 for this fragment), you are currently at step 2 in the workflow. When you consequently use a newer process model and thus replace the old process model, you are at step 2 in the latest workflow, regardless of the rest of the workflow.

CHAPTER 5

Research findings and expert evaluation

Throughout our study, different difficulties affected the ability to maximize the effectiveness of method enactment. In each research, certain occurrences are inevitable but they can usually be resolved rather quickly. Still, this does not take away the fact that the research approach sometimes needs to be altered or extended to fit the research goal. In our research, the modelling freedom of Process Deliverable Diagrams was particular an obstacle for achieving sound results because lots of fragments differed from each other. Therefore, we had to develop the mechanism in such a way that it is aligned with the actual modelling rules. Also, the usage of these diagrams in the CASE tool proved often a complex undertaking. We encountered different problems with respect to the used fragments and the usage of our selected CASE tool Jira. The difficulties affected the creation of our tailor-made solution and sometimes we questioned to what extent some research results could be generalized to other CASE tools. Still it should not be forgotten that each research result is a step forward towards an enhanced solution and that a lot of results and useful information can be derived from such a study.

This chapter will elaborate on the findings during the execution of this research. By not only taking our perspective into account, we also conducted expert evaluations with product managers in order to elicit their opinions and thoughts on our proposed solution in the form of an enactment mechanism. In reality, they will become the end-user of our presented approach. First, we will elaborate on our research findings with regard to process deliverable diagrams as fragments and their evolutionary role in this research. Next, we elaborate on the obstacles we encountered with Jira and the generalization of the research results to other CASE tools. Also, some words on data awareness with regard to the CASE tool are addressed. Finally, results are drawn from the expert evaluations.

5.1 Research findings

The following sub sections elaborate on our personal findings and encountered obstacles during the execution of this research.

5.1.1 Standardization of Process Deliverable Diagrams

Throughout our research, we requested different PDDs from different engineers to validate our developed programming code. During this quest however, we noticed that almost every PDD differed from another. PDDs receive a lot of attention in literature and numerous attempts have been made to standardize its format (e.g. the handbook of Weerd & Brinkkemper, 2008) but unambiguous modelling rules and notations apparently still lack. The attempt by Vlaanderen (2010) to create the objects of a PDD in MetaEdit+ and to add rules and constraints to a PDD is a good first attempt to create modelling standardization in PDDs. Still, even with these objects, rules and constraints there is still too much modelling freedom with regard to its notation. For

instance, we saw PDD's that contained two diagrams in one file, that had two start or stop nodes or that contained a rectangle with an additional textual explanation. Certain objects that are attached to a PDD are easily comprehendible by humans but automation of these PDDs through a system is very difficult to realize due to the presence of unexpected objects in the corresponding XML file. The available objects in MetaEdit+ should therefore be more encompassing by incorporating even more modelling restrictions, notations and rules in order to create valid PDDs. Moreover, only one fragment needs to contain only one complete PDD and not two small PDDs encapsulated in one fragment. Although this seems common-sense, we saw this occurrence numerous times.

5.1.2 The method fragments

In our research, we showed that method fragments can be very dynamic and evolutionary over time. To aid engineers in this process, different types of objects are available to engineers with which they can develop sophisticated fragments. However, the availability of these different types of objects makes the fragments hard to use in other tools. The following subsections elaborate on the fragments of an fragment that caused regularly difficulties during the enactment process.

Open activities, closed activities and sub activities

The process side of a PDD can include open activities, closed activities and sub activities. In a CASE tool however, the usage of open activities and closed activities do not have any meaning because the user of the CASE tool is just transitioning through a status in a workflow (i.e. advancing to the next activity). In PDDs, open and closed activities are typically used to indicate the importance or size of a particular activity by adding a shadowed border for instance. For the CASE tool however, a difference between the model activities does not exist. Modelling open and closed activities are therefore only useful for humans to interpret the size and importance of the respective activity and useless for the CASE tool.

The presence of sub activities in a fragment makes the automation of a fragment rather complicated however. When MetaEdit+ exports a fragment to XML, the XML file does not distinguishes between a regular activity and a sub activity. They both contain the same XML structure and data. This is not surprising because in MetaEdit+, a sub activity is added to the diagram by just adding a regular activity to the model. When the XML file is read by the system, the system cannot determine to which activity a sub activity is connected because it does not contain any outgoing nodes. This is caused by the fact that sub activities are usually used to group a couple of activities and as such, they are not connected to any other activities. Yet, the process of determining the structure of the process flow is influenced by the presence of these sub activities because they are not connected to any other activities. For the system, this could either mean the modeller forgot to add a relationship between two activities or it is a sub activity in the model. Because this scenario would result in a wild guess with a 50 per cent chance, it is recommended to use a different type of symbol to represent a grouping of activities in a fragment (for instance a rectangle as showed in Vlaanderen, 2010). Another solution is to utilize the MetaEdit+ generator and create a custom-made XML exporter. However, because the default exported XML is based on the GOPRR structure, it is questionable whether the creation of

a self-developed exporter will yield more reliable results compared to the thoroughly thought through GOPRR structure.

Forks, joins and decisions

The usage of the different activity nodes that can be modelled in a PDD is limited usable in a CASE tool, taking into account the possibilities of the CASE tool. Jira does allow you to add decision nodes to a workflow and thus we can also use similar decision nodes in our fragments. In this research however, we excluded the usage of decision nodes in our fragments because we had to reproduce the XML string (see section 4.4.3) with our mechanism. Because the creation of this XML string is rather complex and the use of decisions in a workflow does not add to the applicability and feasibility of this research, we leaved them out in our proof of concept. Still, they can be added if desired with some alterations to the programming code and design.

If we take a look at the usage of forks and joins in a fragment, we encountered that the usage of these activity nodes is impossible to implement in a CASE tool. This is mainly caused by the fact that a CASE tool does not support the execution of multiple activities at the same time. Logically, a user can also do one thing at a time. In Jira, the execution of multiple activities at the same time is not supported though the workflow possibilities of the CASE tool are very comprehensive by default. It is therefore questionable if there even is a CASE tool available that supports the usage of forks and joins within a workflow.

Generalizations, associations and aggregations

Concepts within a PDD can have different relations with each other. However, the actual usage of the different relationships that are available in a fragment is difficult to implement in a CASE tool. First of all, the XML file that is produced through MetaEdit+ does not contain any direction that indicates from which concept it starts and to which concept it ends. The XML file only holds information that there is a link between two concepts, but does not add specific details. For instance, when a generalization between two concepts is drawn, you cannot determine from the XML file which concept is the *parent* and which concept is the *child*. The XML file only provides information that there is a link and that is has a specific type; for instance an association. MetaEdit+ determines this by drawing a grid and placing objects in this grid based on coordination's. It is possible to mimic this behaviour but it should be kept in mind that this is rather time consuming and difficult to realize.

Secondly, the multiplicity used in a relationship between concepts is an obstacle. When a concept contains a *one to one* relationship with another concept, it is known that one screen of both concepts is required. However, when a concept has a *one to many* or a *many to many* relationship, it is not known how many screen you need because the amount of concepts is indefinitely. A solution could be to create 100 screens in advance, hoping that the amount will not exceed 100. Obviously such an approach is worthless and contaminates the entire database.

Unfortunately, Jira, in its current form, does not provide any means to support actual relationship between concepts. An only applicable solution can be found in the generalization relationship. When this relationship is used, it is possibility to create a screen based on the concept that is the *parent*. On this screen, you can add field tabs that are based on the *child's* of the respective concept. However, data of *parents* cannot be used in *child's*, thus facing the same problem again.

Other modelling requirements

In addition to the fragments that are present in a fragment, we also encountered other modelling issues that should be taken into consideration when a new method fragment is being produced. If the engineer does not take notice of these modelling concerns beforehand, the CASE tool may cause unexpected behaviour. Therefore, the engineer needs to take the following constraints into account while developing a method:

- An *id* attribute in a concept that refers to a unique *id* is not needed because the CASE tool automatically assigns an *id* to a record.
- Each concept that is present in a fragment should have a unique name to avoid misunderstanding in the CASE tool
- Each attribute that is present in a fragment needs to have a different name. This also applies for the same attribute name in a different concept.
- Each activity name should be unique to avoid misunderstanding in the workflow.
- Each activity should produce at least one concept if the enactment approach is in accordance with this research. If this is not the case, no screens and fields will be visible to the user.
- Usually, a *description* attribute is added to a concept to describe the concept. In a CASE tool however, this is often mandatory so that the user can identify the record. For that reason, a *description* attribute should only be added if this is really necessary.
- An *assignee* attribute in a concept is usually not needed because the CASE tool takes care of this automatically.

It is to say that some of the aforementioned obstacles can also be overcome by adapting the enactment mechanism with the addition of extra rules or code. However, note that this causes a mismatch between the data in the repository of the CASE tool and the actual fragment because the mechanism altered it accordingly. For that reason, this approach is not recommended if both repositories need to be consistent with each other.

5.1.3 CASE tool obstacles

Next to the modelling requirements for a fragment, Jira also caused some obstacles that prevented us from enacting a fragment instantly. Therefore, when also using Jira, the following obstacles should be noticed:

- The Jira service that is running on the machine needs to be stopped first. When the service stopped from running, the fragment can be enacted. After the enactment process, the service could be started again. If we would not restart Jira, all the data from the fragment is not visible in Jira because Jira cashes all its data to the computer memory first when starting the service. Because the fragment data was at that time not yet in database (because the enactment took place later), it is not visible in Jira.
- In some cases, manual intervention is needed after the enactment. For instance if you want to assign a workflow to a project in Jira. Do note however that the GUI of the enactment mechanism is not required to get the process to work. It is in place so that the user can check whether the data of the fragment is analysed in a correct manner.

- Only the concept that is related to an activity is used as a screen. Concepts that are related to other concepts are neglected in the transition screen but can be entered in the global overview screen.

5.1.4 Generalization of the results to other CASE tools

In this research, we selected Jira as our CASE tool as this yielded the most reliable and applicable tool for our research purpose. Although Jira is in fact a bug tracking and project management tool, it proved to be a very good candidate for other purposes due to its high agility and adaptability. Still it should not be forgotten that the results out of this research may not be applicable to other CASE tools. Especially rather limited CASE tools will not provide the same results in contrast to our study.

The selection of the CASE tool also affected the development of the enactment mechanism. Because our mechanism is a tailor-made solution, it is developed with the rationale to enact a fragment in Jira. Because each CASE tool has its own specific characteristics, focus and approach to support the development, it is impossible to create a "one size fits all" solution. In order to make the results applicable to other CASE tools as well, the following requirements play an important role in the selection of a CASE tool:

- The CASE tool should provide the opportunity to access data externally without the CASE tool itself. If this is not available by default, the CASE tool should include a means to connect the CASE tool to an external database. Preferably, the selected CASE tool should be installed locally rather than in the cloud (such as SaaS) because this makes the process of connecting it to an external database manageable and easier.
- The CASE tool should include a means to support the processes side of a PDD through, for instance, a workflow. The workflow should at least include statuses and transitions. For the enactment of more advanced fragments (i.e. fragments that have forks and joins in the process side), the transitions should include support for executing multiple activities at the same time.
- The CASE tool should provide a sufficient amount of possibilities to support the deliverable side of a fragment. In our scenario, we used screens to represent concepts and fields to represent attributes of a concept. A similar approach can be used in other CASE tools but the presence of these possibilities should be investigated beforehand.
- When using a similar approach as we did in our research, the CASE tool should provide a possibility to connect a step in the workflow to one or multiple screen(s). When someone is advancing through the workflow, the appropriated screen can be presented to the user at that moment in the workflow.
- The CASE tool should include the possibility to create different users/groups so that the role in an activity can be implemented properly in the CASE tool.
- The derived data from the fragment that is stored in the database of the CASE tool should be traceable through an identifier. As such it is possible to relate which data belongs to which fragment.

5.1.5 CASE tool data awareness

When we enacted a new fragment in this research, we replaced the old data that was already present in Jira with data from the new fragment. Although this yields good possibilities for

process adaptation and evolution, such an approach is in a real-life scenario usually not recommended due to data complications. Especially when data in the repository of the CASE tool is changed without taking proper actions beforehand, lots of data can become useless or directing to the wrong source. For that reason, we can conclude that there are only two sound solutions for enacting a fragment in an existing CASE tool, namely:

1. The evolution of the fragments over time should only include minor changes, encompassing only a few alterations compared to the previous fragment. Using minor alterations to the fragments lead to foreseeable changes in the CASE tool on which easily can be acted upon.
2. The data in the CASE tool that is associated with the current fragment is finalized first. Then, a new fragment can be enacted in the CASE tool. This solution will not lead to any data complications because prior data in the repository of the CASE tool is always in accordance with its original fragment. Therefore, when a new fragment is enacted, a new *'starting point'* (e.g. a new project) should be created with its corresponding fields and screens.

Especially when major changes are made to a fragment at once, the CASE tool may cause either unexpected behaviour or a major loss of data which can be very risky if someone forgot to make a back-up made beforehand.

5.2 Expert evaluation

The solution that we presented in this research is especially useful for organizations that already use Jira or organizations that need support with the (requirements) management of their software product; may it either be services or products. Within organizations, the success of any product depends on the skills and competence of the product manager (Ebert, 2006). He or she is actually a mini CEO that is responsible for the entire lifecycle of the companies' software product. For that reason, we approached different small-scale and large-scale IT-organizations to ask whether they had product managers in place and if they would help us in evaluating our developed mechanism. The rationale behind the evaluation of the mechanism is to get a good understanding of how our presented solution is perceived by product managers because they will be the end-users. Furthermore, we wanted to obtain opinions, remarks and other concerns regarding the developed mechanism which we could use as input for a subsequent version of the mechanism.

5.2.1 Interview set-up

To derive the appropriate information, we conducted five interviews at different IT-companies where each company developed software for a specific industry. In each company, we introduced the rationale of our research and the general idea behind the developed mechanism. To elicit opinions, remarks and any other information about our mechanism, we based our interviews on a semi-structured basis. In this type of interview, there are some questions prepared beforehand but that there is also room for improvisation (Fontana & Frey, 2000). In total, we created 14 questions beforehand (see appendix IV) from which 7 questions are aimed at eliciting information directly related to the companies' current situation. The other 7 questions focus on the mechanism itself and its working. After the 14 questions, time was

available to document any kind of other information that was provided by the product manager, i.e. remaining concerns and/or opinions.

The interview encompassed two parts; a part in which we asked 7 questions directly related to the companies' current situation and a part in which we asked 7 questions regarding the demonstrated mechanism. In between, a demo was shown to the product manager in the form of a video that showed the mechanism in action. During the video, a verbal explanation was provided to make things clearer to the product manager where the video could be stopped at any moment in time if desired. The reason for choosing a video instead of a live demonstration was because Jira and the mechanism were too demanding with regard to CPU and memory resources for our notebook. Therefore we recorded the screen when the mechanism was in action and used this as our video during the interview. The interview constituted two parts because we did not want the demo to affect the answers related to the companies' current situation. Therefore we asked about the current situation in the organization first and then proceeded to the enactment video demonstration. Before we showed the video to the product manager however, we first elaborated on the notion of PDDs and the used fragments so that the product managers were familiar with both concepts. After this explanation, the video was showed and the second part of the interview took place.

5.2.2 Expert evaluation results

The result of the expert evaluation is divided in two sections. First we will address the current method usage within organizations by asking organizations how they currently deal with process adaptation and process evolution. Secondly, we will elaborate on the evaluation of our developed mechanism. In Table 12, details of the interviewed companies are provided. For privacy reasons, the company names have been left out. Also note that the interview answers are related to the main software product the respective company is developing and that answers are hence based on this software product. The used questions for this interview can be found in appendix IV.

Table 12. Details of interviewed companies.

Company	Function interviewee	Amount of employees involved in core product	Core product
A	Product manager	25 (25 total)	Information management software
B	Product manager	±23 (±5300 total)	Pay rolling and Human Resource Management software
C	Head of product	±50 (±2000 total)	Social media
D	Product specialist	±30-150 (±5300 total)	*N.A. (internal researcher)*
E	Product manager	3 (±300 total)	ERP vendor

Organizations' current method usage

Recognition of process evolution

For each organization, we first asked whether they recognized the need for process adaptation and how they currently deal with process adaptation. Each of the interviewed companies agreed on the fact that process adaptation is recognized throughout the organization or department. Each company struggled in the beginning with their business process(es) within their

company/department and each company reinvented the wheel numerous times before a standard process was in place. The process was especially in the beginning continuously updated based on new information. Over time however, the process was used less often than originally intended; mainly caused due to the high level of detail in the model.

"We do see the need for continuous process adaptation but we usually lack proper actions"

High level processes

Four out of five companies has process models in place, mostly at a high level of detail. One company did not have any model in place due to its small size. Over time, each company shifted from having a very detailed process model to a more high-level process. Company A (see Table 12) works without having any process model in place and processes are performed in an ad-hoc manner. Medium-scale and large-scale organizations (approximately 300 employees or above) only have a high-level overview of their processes. In such a process, each person is responsible for a particular part in the process and they rather use the model as phases during the entire lifecycle of the software product. There are a lot of (external) influences that affect the flow of the process and hence, a detailed process model becomes easily cluttered or too complex.

SCRUM usage

We also noticed that two out of four companies that had models in place utilized SCRUM (an agile software development method) to deal with the management of their software. For each sprint, the same roles and tasks are assigned as the previous sprint. Based on the interview results, we saw that little detailed processes are in place and that companies are rather using SCRUM in conjunction with high-level processes.

The enactment mechanism

For the evaluation of our enactment mechanism, we asked product managers about their opinions regarding the enactment mechanism. We showed a video that illustrates how the enactment mechanism works, its primary goal and its advantages to overcome process adaptation and evolution. In total, seven questions were asked (see appendix IV).

Beneficial

Out of five companies, one company agreed upon the fact that a lot of companies that are using Jira can benefit from the approach. It took a lot of time to configure Jira but with the mechanism, the entire configuration of Jira can be realized with one click. He described it as a major advantage with which we could even make money if we develop it in a correct manner. Yet he also noted that the solution should be kept simple. Two companies understood the rationale behind the approach, but both concluded that the process is in reality more complicated and has more (external) influences that may affect the process flow. The used fragments in the research show a linear process, but in fact, the process is often much more complex. We recognize this indeed as a deficiency in the

"I experienced Jira as too complicated, too comprehensive and too difficult to configure"

current mechanism. Both companies concluded that the process model is usually different from the actual working situation and more complicated than reflected in the demo. Therefore, they propose a better elaboration of the current concept making it more comprehensive with at least the inclusion of decisions in the workflow.

> *"If you develop this is a correct manner, you can even make money with it"*

Too rigid

One company did not like the solution at all because they found it *"too rigid"*. The mechanism should include more flexibility and agility in the process model; especially with regard to the process side. They argued that the approach adapts working habits of people to fit your IT. They believe it should be the other way around and that IT should be adapted to fit the working habits of people.

"Start and see what happens" implementation approach

We also asked how product managers would implement the approach within their department/organization and how they would inform people about the new approach. The interviewed companies that are mainly using SCRUM for the management of their software product would simply provide the solution to the employees and ask for opinions and thoughts on the approach and evaluate this. Basically, they would use a "start and see what happens" approach, similar to the SCRUM principles where you usually start developing first and evaluate the result afterwards. Three other interviewed companies would convince people that the approach is a step forward towards the maturation of the company and that the implementation should be thoroughly ingrained first.

Increase of resistance

Furthermore, we asked whether companies would expect resistance when the demonstrated approach would be implemented in the organization. Based on the interviews, small-scale organizations would probably expect a lot of resistance from employees due to its rigidity. The fact that employees should work according to a predefined process decreases the flexibility within organizations because small-size companies are more collaborating "on-the-fly" (i.e. ad-hoc) with each other. Additionally, one company argued that the approach we demonstrated would lead to *"documenting information because the tool says you need to document it"*. However, if the process model is evaluated numerous times over time, such a scenario can easily be prevented by adapting the model to fit the needs for the organization or department.

Data deletion awareness

In the interviews, we also asked the product managers whether they missed something in the demonstration that has been overlooked. One product manager indicated that the deletion of method fragments in a subsequent fragment should be handled with caution. Deleting data from the repository is often not desired because prior data can always be useful for organizations in the future. This also applies for a situation where data is moved from one location to another location in a subsequent fragment. *Is the data in such a scenario still present?* Hence a suggestion

was made to include an option to choose between deleting or (temporary) hiding data from/in the repository.

Lack of comprehensive features

Overall, the feasibility of the mechanism was questioned in its current form, mostly caused by its lack of comprehensive features. Questions rose such as:

- Who is responsible for the process model(s)?
- Who keeps the process model(s) up-to-date?
- What if activities need to be carried out simultaneously?
- What if the process is iterative?

The questions mentioned above all have to deal with the mechanism in its current form. Especially companies that already have Jira in place can benefit from the proposed solution, but it should be noted that the current mechanism should include more functionality before it becomes a useful tool. Another notable remark is that all the suggestions were directed at the process side of a fragment. Therefore, the mechanism needs to incorporate much more flexibility with regard to the process side of a fragment and its operationalization into the CASE tool.

Conclusions

Finally, we asked whether the product managers would use the approach we showed in the demonstration. Four out of five interviewed companies concluded that they would only use such an approach if it:

"The demonstrated approach can be very useful but should stay simple"

- stays simple;
- encompasses more features for workflow support; especially carrying out activities simultaneously or making decisions;
- can be used as a means to show the transparency of their (internal) processes towards the customer(s).

One company did not felt the need to use such a solution due to its rigidity and small amount of features. Also, the deliverable side of the PDD was perceived as too complicated to model and requires a good understanding of UML; which not every company had to a sufficient degree.

CHAPTER 6

Discussion and limitations

In this study, we established a different viewpoint of how models can be useful if they are being developed within organizations. Today there are more than hundred types of different models available that all aim to provide insight in (business) processes within organizations. The making of these process models however tends to absorb a lot of time and resources and in the end, they are often rarely used.

In this research, we took it one step further by using the process models as input for other tools. Rather than just developing another model that supports process adaptation and evolution, we showed an approach of how the presence of models can add value to the company by using the data of the process model to affect software engineering tools that support companies in the making of their software. Regular process models often emphasize how processes should be carried out within an organization, but usually do not take the results of the processes into account. Because the results are as vital as the processes that are carried out, we adopted both viewpoints in our diagram to show how important it is to be aware of these deliverables. In fact, they are providing what a company is actual looking for.

In our research we developed our own tailor-made enactment mechanism because this proved to be the most practical solution to accomplish our research goal. Nevertheless, it should be noted that the enactment mechanism in its current form is specific to Jira only. For that reason the current enactment mechanism is not generalizable to other CASE tools. During our quest to find a suitable and usable CASE tool, we noticed that each CASE tool had its own specific characteristics, focus and approach to support the software development process. Due to the major differences between existing tools, it is impossible to create a mechanism that includes support for each CASE tool. The abstraction level would be tremendously high and the solution would become so complex that such a solution is impossible to realize. Nonetheless, this research showed that with the right tools, the enactment of a fragment is feasible as long as it is focused towards a specific research goal.

The enactment mechanism

Jira plugin

If we reflect on our custom-made mechanism and our research findings, the mechanism can be very useful to medium-scale and large-scale companies that already have Jira in place. In order to get companies aware of the advantages of the developed mechanism, we searched for possibilities to increase this awareness. During this research we noticed that Jira offers a great amount of possibilities for extending its elementary functionality. One of the available functions of Jira is that it allows you to create your own custom-made plug-ins and use them in Jira. This possibility yields a great opportunity for providing a first start to an enactment mechanism

plugin for Jira. If the enactment mechanism in its current form can be transformed to Java programming code (note that both notions are based on the Object Oriented Programming principle), a Java plug-in can be established for Jira. This plug-in can be uploaded to the Atlassian website whereby it will become available to all the users of Jira worldwide. Every organization that is concerned with process evolution and adaptation can thus be supported with the developed plug-in. Before this can be achieved however, the current mechanism should incorporate more basic functionality and advanced features to make it applicable for such a research goal. Also, the deliverable side of a PDD was perceived as complicated. Therefore, using simple PDDs as fragments had the product manager's preference.

Easiness of switching requirement prioritization techniques

Another notable advantage with the current mechanism is the easiness of which a different prioritization technique can be selected when a company is dealing with requirements. The mechanism, in its current form, is able to cope with simple requirement prioritization techniques and use these techniques in the CASE tool. If a company wants to change its prioritization technique, a new prioritization technique can be created or a previous prioritization technique can be altered in the new fragment. When this data is used in the CASE tool, organizations can easily switch to different prioritization techniques with the right data transformation. In its current form, the mechanism can only deal with simple prioritization technics and more advanced prioritization techniques such as AHP (Saaty, 1980) might become available. However, this only applies if the CASE tool is able to provide the necessary data and make the right calculations before it presents data on the screen. In Jira, this is not yet possible.

Prior CASE tool data awareness

An important concern that should not be overlooked is the presence of prior data in the repository of the CASE tool. In this research, we could easily alter existing data or add new data to the repository of the CASE tool with our mechanism. However, the deletion of data causes much more difficulties because destroying data means that it cannot be restored anymore; a situation that is usually not preferred. Our selected CASE tool contains a rather extensive amount of features to overcome this problem by, for instance, hiding prior data. Most likely, the most appropriate solution is the possibility to hide data instead of deleting it. Hiding data means that it is not visible to the regular user anymore, but still remains in the repository of the CASE tool. In this way, data can always be restored or consulted at any moment in tool. However, if the selected CASE tool does not include an option to hiding data, a data transformation process should be examined thoroughly before a similar approach is employed.

Process Deliverable Diagrams

In existing literature, a lot of information can be found regarding Process Deliverable Diagrams. Although the diagrams are discussed rather frequently, a perceptible operationalization of these diagrams not yet exists. They can be seen as yet another diagram to process modeling. In this research, we illustrated an approach where the usefulness of PDDs can be increased by operationalizing them as a useful tool in other software engineering tools. With the use of an enactment mechanism, we established a tangible elaboration of PDDs and we illustrated the

importance of the deliverable side of the diagram. In fact, the deliverables provide the company with the necessary data.

Lack of unambiguous modeling rules

Based on the results of our study, we identified obstacles and limitations with regard to Process Deliverable Diagrams and their modeling notation. Although PDDs are described in great detail in existing literature, unambiguous modeling rules are hard to derive because different scientific papers sometimes contradicted each other. This variety of modeling notations in different scientific papers resulted in different types of PDDs. Uniformity with regard to PDDs is not yet achieved, making it hard to develop a mechanism that includes every rule from every source. The study we presented in this research identified deficiencies and obstacles of using the data of PDDs as input for an automated process. This information can be used to create more unambiguous modeling rules and to tackle the problems in the future that we encountered in our research. Altogether; the results that can be derived from this study can be very useful to increase the efficiency and applicability of future PDDs and the process of automating data within a PDD.

Research method

Utilized research approach

If we look back on our utilized design science research approach (i.e. the approach defined by Peffers *et al.,* 2008), we found that the approach provides sufficient handles for guiding the creation of an artifact. Still, the approach is rather a high level overview of different steps that need to be carried out in an iterative manner. Due to this level of granularity, detailed steps that describe which actions are needed to achieve successful results are absent. In our research, this was not a major problem because we had sufficient programming skills and knowledge to software development. However, for those that lack this knowledge, the design science research may be too superficial to achieve useful results.

Artifact development

A major drawback in our approach was the development of the first version of the enactment mechanism, which was not thought thoroughly through. In the first version, a lot of deficiencies arose mainly caused by the poor development design. As with a lot of programming code, the code becomes (too) cluttered over time where you need to make the decision: do I continue with this or do I start all over again? This problem also arose in our research and hence we reassessed the entire development design. Eventually we made the decision to redo the entire enactment mechanism and to base it on the Model-View-Controller (MVC) design pattern. Aware of the fact that this decision affected our entire development cycle and research approach, we still believe this was the right decision as this yielded the most practicable and reliable results to our second version of the mechanism. Software development is typically an iterative process where the artifact that you are creating is continuously updated based on previous information. Due to the major amount of possibilities to achieve similar results, it should be noted that our development design is just one approach to method enactment. A lot of other approaches can therefore yield similar results.

Interview approach

Looking back at the evaluation of our mechanism through product managers, using a semi-structured interview was a good approach for eliciting opinions and thoughts. During the interviews, we noticed that each product manager had a different opinion related to the mechanism. If we would use a structured interview or a questionnaire where product managers would not have the possibility to express thoughts and opinions on the mechanism, a lot of valuable information for future research would remain unused.

Limitations

During the execution of our research, we encountered some limitations that influenced our research direction and the development of the enactment mechanism. Per selected tool in our research, we describe the different limitations that arose below.

CAME tool

1. *Reading of the XML file*
   The reading of the XML file through the mechanism is based on the GOPRR structure that is produced by MetaEdit+. As a result, the mechanism can only extract data from the XML file if this file is in compliance with the GOPRR structure. If another structure is used (for instance by using another program to create the fragments), the mechanism is not able to derive the right data from the fragment.

CASE tool

2. *Jira specific*
   The enactment mechanism that we developed in this research is specific to Jira only. The development rationale of the mechanism is based on the table structure of Jira and the affected tables when inserting or modifying data in the repository of Jira. Because each CASE tool differs in structure from another, the mechanism cannot be used for other CASE tools.
3. *Version specific*
   The mechanism is developed for Jira version 4.4.4 and is therefore based on the structure and working of this version. At this moment, version 5.1 of Jira is already released and if the table structure of Jira changed during version 4.4.4, the mechanism may cause problems while inserting or modifying the data in the repository of Jira. Therefore, caution should be taken if a different version of Jira is used.

Enactment mechanism

4. *WAMP environment*
   The approach employed in this research requires an installation of the Apache environment on the machine in order to parse the PHP code that is used for the creation of the mechanism. Without a PHP parser, the enactment mechanism cannot be parsed and hence not store any data in the repository of Jira. Furthermore, an external database (in this research a MySQL database was used) is required in order to connect Jira to this

database and to execute the queries on this database. In case another database type is used, the queries need to be changed to fit this database type.

5. *One method can be used only*
   The mechanism in its current form is constrained by the usage of fragments that apply to one specific evolutionary process. If there are multiple fragments used from multiple processes, Jira will cause unexpected behaviour if similar concepts, attributes or activities are used in the different processes. For that reason it is recommended to use the proof of concept that we developed in this research for one process only. Also, the usage of the same concept names, attribute names and activity names throughout a fragment may cause different results.
6. *Omission of roles*
   The different roles that can be modelled in a fragment have been left out in the mechanism. By default, Jira includes an option to create different roles and this can be added to the mechanism if desired.
7. *Logical flow only*
   The proof of concept is only capable of determining and enacting a logical flow in a fragment. The usage of decisions, forks, joins and sub activities in a fragment will cause problems while reading the XML. For that reason, these activity nodes cannot be included in a fragment yet.

To overcome the above mentioned concerns and to meet the requirements for adopting the mechanism as a suitable tool for organizations that are using Jira, we argue that a more comprehensive mechanism is needed that takes the aforementioned limitations into account. If the limitations can be mitigated in a future release, a powerful and easy to use tool becomes available.

CHAPTER 7

Conclusion and future research

In this research, we attempted to provide an answer to the following research question:

"How can method increments be employed in software engineering tools and allow for incremental method enactment?"

To answer this research question, this study demonstrated an example of how method fragments can be implemented in a software engineering tool with a custom developed *enactment mechanism.* The enactment mechanism that we developed in this study consists of four main phases, namely an *analyse, populate, transform* and *store* phase and is developed with the programming language PHP in conjunction with MySQL. The *analyse* phase is in place to analyse the raw method fragment data in the form of XML and determines whether the fragment is complete and valid. The *populate* phase populates this raw fragment data and extracts the necessary data that is needed for the software engineering tool. This data is temporary stored in a structured manner. The *transform* phase will transform the temporary data to a standard that is in compliance with the software engineering tool. Finally, the *store* phase will store all the elicited data from the fragment in the repository of the software engineering tool. When the four phases are executed successfully in a chronological order, the lay-out of the corresponding software engineering tool is tailored to the created fragment. When the enactment mechanism is invoked after each created fragment, incremental method enactment is achievable in the corresponding software engineering tool.

The results in this research provide a first start to the problems that some medium-scale or large-scale organizations face. Certain organizations usually tend to create different types of (process) models with the aim to support their software development process. In most cases however, the organizations fail to comply to these process models because the models often remain unused. Yet, the resources in terms of time and money have already been devoted to the development of these models.

In this research, we provided a mechanism that can support organizations to employ their models in an already employed software engineering tool. We used the developed process models as input for the engineering tool and showed an approach of how the presence of models can add value to the company by using the data of the process model to affect software engineering tools that support companies in the making of their software. Especially companies that are already using Jira can benefit from the demonstrated approach. Jira was generally perceived as complex, ponderous and difficult to configure, but with the use of the demonstrated mechanism, everyone is able to configure Jira without having a profound understanding of Jira itself. This was recognized by product managers as a major advantage of the mechanism. It should be noted however that PDDs were perceived as rather difficult and somewhat complex

on the deliverable side. Therefore, using simple PDDs is more promising. Also, the enactment mechanism, in its current form, is perceived as not comprehensive enough to act as a useful mechanism. It should incorporate more valuable functionality, especially with regard to available options within a workflow. Support for decisions, forks and joins in a workflow is thus desired.

The enactment of a fragment in a software engineering tool can be twofold. The mechanism can either create an entire new workflow in the software engineering tool or it can replace and delete existing data in the software engineering tool. In this research, we focused on the second approach. We demonstrated how the lay-out and data in Jira that corresponds with a fragment can be replaced based on a newer fragment. In this way, the lay-out is incrementally changed while the end-users are using the engineering tool. However, the replacement of this lay-out and its corresponding data may cause unexpected or unforeseeable behaviour in the software engineering tool; especially when major changes are made in a newer fragment. Also, data can be lost if this has been omitted in a newer fragment. Therefore, with respect to the method fragments, it is recommended to either

1. keep the changes between different fragments minor as this will lead to foreseeable changes in the CASE tool on which can easily acted upon; or
2. to create an entire new workflow, issue type and the corresponding screens and fields in the CASE tool.

Especially the latter solution proved to be the most practical because this mitigates any problems that arise with existing data in the repository of the CASE tool. Changing the lay-out incrementally was not accounted as a practical and usable solution according to our expert evaluation. They indicated that they wanted to keep prior data rather than destroying it. For that reason, each product manager referred to option 2 as the best solution. If we reflect on our mechanism, we also agree on the fact that option 2 will provide the most reliable results and will reduce the amount of obstacles afterwards. Therefore, incremental method enactment was only desired if it does not affect previous data that was already in the repository of the CASE tool. Based on the results of our study, we are able to conclude that incremental method enactment is feasible but that the mechanism should include additional functionality that allow product managers to meet their own preferences. Also we showed that CASE tools can be very flexible and adaptable to fit other research purposes, even though they are not specifically made for certain purposes.

Future research

In order for the mechanism to become a useful tool, the current mechanism needs more basic functionality. It should be extended by incorporating support for decisions, joins and forks in a workflow, although this is partially influenced by the selected CASE tool. Also, it is recommended basing a future mechanism on the Java programming language because this makes it easier to use the mechanism as a Jira plug-in. Furthermore, opinions and remarks of product managers that we addressed in our research should be taken into account in order for the mechanism to become useful and convenient to use. Finally, it is recommended to extend the current mechanism with the inclusion of a data transformation option rather than deleting data.

References

Aalst, W. M. P. van der, Hofstede, A. H. M. ter, & Weske, M. (2003). Business Process Management: A Survey. In M. Weske (Eds.), *Lecture Notes in Computer Science: Vol 2678* (pp. 1-12). Heidelberg, Germany: Springer. doi: 10.1007/3-540-44895-0_1

Alavi, M. (1993). Making CASE an Organizational Reality – Strategies and New Capabilities Needed. *Information Systems Management, 10*(2), 15-20. doi: 10.1080/10580539308906923

Amelunxen, C., Klar, F., Königs, A., Rötschke, T., & Schürr, A. (2008). Metamodel-based Tool Integration with MOFLON. *Proceedings of the 30th International Conference on Software Engineering (ICSE'08),* New York, USA, 807-810. doi: 10.1145/1368088.1368206

Arni-Bloch, N. (2005). Towards a CAME Tools for Situational Method Engineering, Doctoral Symposium, *International Conference INTEROP-ESA*, Geneva, Switzerland, February 2005.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). The unified modeling language user guide. Redwood City, CA: Addison Wesley Longman Publishing Co.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., & Cowan, J. (2006). Extensible Markup Language (XML) 1.1 (2nd ed.). *W3C recommendation.* Retrieved March, 20, 2012 from http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf.

Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information and Software Technology, 38*(4), 275-280. doi: 10.1016/0950-5849(95)01059-9

Brinkkemper, S., Saeki, M., & Harmsen, F. (1999). Meta-modelling based assembly techniques for Situational Method Engineering. *Information Systems, 24*(3), 209-228. doi: 10.1016/S0306-4379(99)00016-2

Burmester, S., Giese, H., Niere, J., Tichy, M., Wadsack, J. P., Wagner, R., Wendehals, L., & Zündorf, A. (2004). Tool Integration at the Meta-Model Level: the Fujaba Approach. *International Journal on Software Tools for Technology Transfer (STTT), 6*(3), 203-218. doi: 10.1007/s10009-004-0155-8

Dahanayake, A. (2001). *Computer-Aided Method Engineering: Designing CASE repositories for the 21st Century.* Hershey: Idea Group Publishing.

Demarco, T. (1979). *Structured Analysis and System Specification.* New Jesery: Yourdon-Press.

Ebert, C. (2006). The impacts of software product management. Journal of Systems and Software, 80(6), 850-861. doi: 10.1016/j.jss.2006.09.017

Fontana, A., & Frey, J. H. (2000). The interview: from structured questions to negotiated text. In N. K. Denzin & Y. S. Lincoln (Eds.), Handbook of qualitative research (2nd ed.) (pp. 645–672). Thousand Oaks, CA: Sage.

France, R., & Rumpe, B. (2005). Domain specific modeling. *Software and Systems Modeling, 4,* 1-3. doi: 10.1007/s10270-005-0078-1

Gupta, D., & Prakash, N. (2001) Engineering Methods from Method Requirements Specifications. *Requirements Engineering, 6*(3), 135-160. doi: 10.1007/s007660170001

Harmsen, A. F. (1997). Situational Method Engineering (Doctoral Dissertation). Moret Ernst & Young Management Consultants, Utrecht.

Harmsen, F., & Brinkkemper, S. (1995). Design and implementation of a method base management system for a situational CASE environment. *Proceedings of the ASPEC'95 Conference*, Brisbane, Australia, 430-438. doi: 10.1109/APSEC.1995.496992

Harmsen, F., Brinkkemper, S., Oei, J.L.H. (1994). Situational method engineering for information system project approaches. In: T. Olle and A. Verrijn Stuart, Editors, Methods and Associated Tools for the Information Systems Life Cycle, *Proceedings of the IFIP WG8.1 Working Conference CRIS'94,* (pp. 169-194). Amsterdam, The Netherlands: North-Holland.

Henderson-Sellers, B., & Ralyté, J. (2010). Situational Method Engineering: State-of-the-Art Review. *Journal of Universal Computer Science, 16*(3), 424-478. doi: jucs-016-03-0424

Henderson-Sellers, B., Gonzalez-Perez, C., Ralyté, J. (2008). Comparison of Method Chunks and Method Fragments for Situational Method Engineering. *19th Australian Conference on Software Engineering (ASWEC 2008)*, Perth, Australia, 479-488. doi: 10.1109/ASWEC.2008.4483237

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75-105.

Heym, M., & Osterle, H. (1992). A semantic data model for methodology engineering. *Proceedings of the 5th International Workshop on Computer-Aided Software Engineering,* Montreal, Canada, 142-155. doi: 10.1109/CASE.1992.200144

ISO. (1990). ISO-IEC 10027: Information technology - Information Resource Dictionary System (IRDS) - Framework, ISO/IEC International standard.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Massachusetts: Addison-Wesley.

Jarzabek, S., & Huang, R. (1998). The Case for User-Centered CASE Tools. Communications of the ACM, 41(8), 93-99. doi: 10.1145/280324.280338

Karsai, G., Lang, A., & Neema, S. (2003). Tool Integration Patterns. *Workshop on Tool Integration in System Developement, ESEC/FSE,* Helsinki, Finland, 33-38.

Karsai, G., Lang, A., & Neema, S. (2005). Design patterns for open tool integration. *Software and Systems Modeling, 4*(2), 157-170. doi: 10.1007/s10270-004-0073-y

Kelly, S., Lyytinen, K., & Rossi, M. (1996). MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment. In P. Constantopoulos, Y. Vassiliou, & J. Mylopoulos (Eds.), *Lecture Notes in Computer Science: Vol 1080* (pp. 1-21). Heidelberg, Germany: Springer. doi: 10.1007/3-540-61292-0_1

Kock, N., Gray, P., Hoving, R., Klein, H., Myers, M., & Rockart, J. (2002). IS Research Relevance Revisited: Subtle Accomplishment, Unfulfilled Promise, or Serial Hypocrisy?. *Communications of the Association for Information Systems, 8*(23), 330-346.

Könings, A., & Schürr, A. (2006). Tool Integration with Triple Graphs Grammars – A Survey. *Electronic Notes in Theoretical Computer Science, 148*(1), 113-150, doi: 10.1016/j.entcs.2005.12.015

Kornyshova, E., Deneckère, R., & Salinesi, C. (2007). Method Chunks Selection by Multicriteria Techniques: an Extension of the Assembly-based Approach. In J. Ralyté, S. Brinkkemper, & B. Henderson-Sellers (Eds.). *Proceedings of the International Federation for Information Processing WG 8.1 Working Conference: Vol. 244. Situational Method Engineering: Fundamentals and Experiences* (pp. 64-78). Berlin, Germany: Springer. doi: 10.1007/978-0-387-73947-2_7

Kottemann, J. E., & Konsynski, B. R. (1984). Dynamic Metasystems for Information Systems Development. *Proceedings of the 5th International Conference on Information Systems,* Tucson, Arizona, USA, 187-204.

Kramler, G., Kappel, G., Reiter, T., Kapsammer, E., Retschitzegger, W., & Schwinger, W. (2006). Towards a Sematic Infrastructure Supporting Model-based Tool Integration. *Proceedings of the 2006 International Workshop on Global Integrated Model Management (GaMMa'06),* New York, USA, 43-46. doi: 10.1145/1138304.1138314

Kumar, K., & Welke, R. J. (1992). Methodology Engineering: a proposal for situation-specific methodology construction. In: W. W. Cottermand, & W. A. Senn (Eds.), *Challenges and strategies for research in systems development* (pp. 257-269). New York: John Wiley & Sons, Inc.

March, S. T., & Smith, G. F. (1995). Design and Natural Science Research on Information Technology, *Decision Support Systems, 15*(4), 251-266. doi: 10.1016/0167-9236(94)00041-2

McLure, C. (1989). The CASE experience. *Byte, 4*(14), 235-244.

Niknafs, A., & Ramsin, R. (2008). Computer-Aided Method Engineering: An Analysis of Existing Environments. In Z. Bellahsène, & M. Léonard (Eds.), *Lecture Notes in Computer Science: Vol 5074* (pp. 525-540). Heidelberg, Germany: Springer-Verlag. doi: 10.1007/978-3-540-69534-9_39

Object Management Group. (2004). *UML 2.0 superstructure specification* (Technical Report ptc/04-10-02).

Object Management Group. (2011). *OMG Meta Object Facility (MOF) Core Specification* (formal/2011-08-07).

Oinas-Kukkonen, H. (1996). Debate Browser - an Argumentation Tool for MetaEdit+ Environment. *Proceedings of the 7th European Workshop on next generation of CASE Tools (NGCT'96)*, Heraklion, Greece. 77-86.

Orlikowski, W. J., & Iacono, C. S. (2001). Research Commentary: Desperately Seeking the "IT" in IT Research – A call to Theorizing the IT Artifact. *Information Systems Research, 12*(2), 121-134. doi: 10.1287/isre.12.2.121.9700

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems / Winter, 24*(3), 45-77. doi: 10.2753/MIS0742-1222240302

Premkumar, G., & Potter, M. (1995). Adoption of Computer Aided Software Engineering (CASE) technology: an Innovation Adoption Perspective. *Diffusion of Technological Innovation, 26*(2-3), 105-124. doi: 10.1145/217278.217291

Ralyté, J. (1999). Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base. *Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99),* Florence, Italy, 1-12. doi: 10.1109/DEXA.1999.795184

Ralyté, J., & Rolland, C. (2001). An Approach for Method Reengineering. *Proceedings of the 20th International Conference on Conceptual Modelling, ER2001*, Yokohama, Japan, 471-484.

Ralyté, J., Deneckère, R., & Rolland, C. (2003). Towards a generic model for situational method engineering. In J. Eder, & M. Missikoff (Eds.), *Lecture Notes in Computer Science, Vol. 2681* (pp. 95–110). Heidelberg, Germany: Springer-Verlag. doi: 10.1007/3-540-45017-3_9

Rolland, C. (1997). A primer for Method Engineering. *Proceedings of the INFormatique des Organisations et Systèmes d'Information et de Décision (INFORSID'97),* Toulouse, France,

Rolland, C., Plihon, V., & Ralyté, J. (1998). Specifying the reuse context of scenario method chunks. *Proceedings of the 10th Conference on Advanced Information Systems Engineering*, Pisa, Italy, 191-218. doi: 10.1007/BFb0054226

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lonrensen, W. (1991). *Object-Oriented Modeling and Design.* New Jersey: Prentice-Hall.

Russo, N. L., Wynekoop, J. L., & Walz, D. B. (1995). The use and adaptations of system development methodologies. *Proceedings of the Information Resources Management Association International Conference,* Atlanta, USA.

Saaty, T. L. (1980). *The Analytical Hierarchy Process*. New York: McGraw-Hill.

Saeki, M. (1998). Method Engineering. In P. Navrat, & H. Ueno (Eds.), *Knowledge-Based Software Engineering*, IOS Press.

Saeki, M. (2003). CAME: The First Step to Automated Method Engineering. *Workshop on Process Engineering for Object-Oriented and Component-Based Development,* Anaheim, USA, 7-18.

Saeki, M., Iguchi, K., Wenyin, K., & Shinohara, M. (1993). A Meta-Model for Representing Software Specification & Design Methods. *Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process*, Amsterdam, The Netherlands, 149-166.

Salminen, A., & Tompa, F. (2011). Why use XML?. In A. Salminen, & F. Tompa (Eds.), *Communicating with XML* (pp. 69-91). Heidelberg, Germany: Springer. doi: 10.1007/978-1-4614-0992-2_3

Seligman, L., & Roenthal, A. (2001). XML's Impact on Databases and Data Sharing. *IEEE Computer Society, 34*(6), 59-67. doi: 10.1109/2.928623

Si-Said, S., Rolland, C., & Grosz, G. (1996). MENTOR: A Computer Aided Requirements Engineering Environment. In P. Constantopoulos, Y. Vassiliou, & J. Mylopoulos (Eds.), *Lecture Notes in Computer Science: Vol 1080* (pp. 22-43). doi: 10.1007/3-540-61292-0_2

Tekinerdoğan, B. (2000). *Synthesis-Based Software Architecture Design* (Doctoral Dissertation). University of Twente, Twente, the Netherlands.

Tolvanen, J. (2004). MetaEdit+: domain-specific modelling for full code generation demonstrated [GPCE]. *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA'04)*, New-York, USA, 39-40. doi: 10.1145/1028664.1028686

Tolvanen, J., & Rossi, M. (2003). MetaEdit+: defining and using domain-specific modeling languages and code generators. *Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages and applications,* Anaheim, USA, 92-93. doi: 10.1145/949344.949365

Vlaanderen, K. (2010). Improving Software Product Management Processes: a detailed view of the Product Software Knowledge Infrastructure. (Master thesis). Utrecht University, Utrecht, the Netherlands.

Wasserman, A. I. (1990). Tool Integration in Software Engineering Environments. *Proceedings of the International Workshop on Software Engineering Environments,* Chinon, France, 137-149. doi: 10.1007/3-540-53452-0_38

Weerd, I. van de, & Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. In M. R. Syed, & S. N. Syed (Eds.), Handbook of Research on Modern Systems Analysis and Design Technologies and Applications (pp. 38-58). Hershey: Idea Group Publishing.

Weerd, I. van de, Brinkkemper, S., & Versendaal, J. (2007). Concepts for Incremental Method Evolution: empirical exploration and validation in Requirements Management. In J. Krogstie, A. Opdahl, G. Sindre (Eds.), *Lecture Notes in Computer Science: Vol. 4495* (pp. 469-484). Heidelberg, Germany: Springer.

Weerd, I. van de, Brinkkemper, S., & Versendaal, J. (2010). Incremental Method Evolution in global Software Product Management: a Retrospective Case Study. *Information and Software Technology, 52*(7), 720-732. doi: 10.1016/j.infsof.2010.03.002

Weerd, I. van de, Brinkkemper, S., Souer, J., & Versendaal, J. (2006). A situational implementation method for web-based content management system-applications: method engineering and validation in practice. *Software Process: Improvement and Practice, 11*(5), 521-538. doi: 10.1002/spip.294

Weske, M. (2007). *Business Process Management: concepts, languages, architectures*. New York: Springer-Verlag.

Appendices

Incremental Method Enactment in Software Engineering Tools

MBI Master Thesis

IKU-3555135

Geurt van Tuijl

25 August 2012

APPENDIX I

Used Method Increments

This appendix shows the five method fragments that were used in this research. The fragments show an evolutionary approach of a requirements prioritization process and are similar to the fragments in Weerd, Brinkkemper and Versendaal in 2010. An explanation of these fragments can therefore be found in Weerd *et al.* (2010).

Because this research is constrained by the usage of sub activities and decisions in the process side of the diagram, they are omitted in the fragments that were subject to this research. The usage of the colour overlays in the fragments is as follows:

A method fragment or part thereof is inserted or modified.

A method fragment or part thereof is deleted.

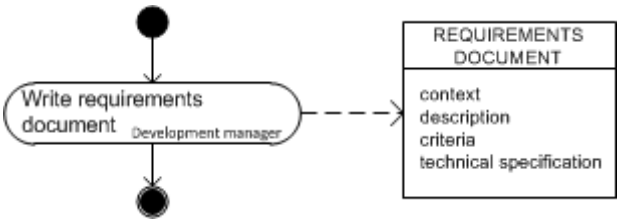


Figure A1.1. Fragment 1 of the requirement prioritization process in *Weerd et al.* (2010).

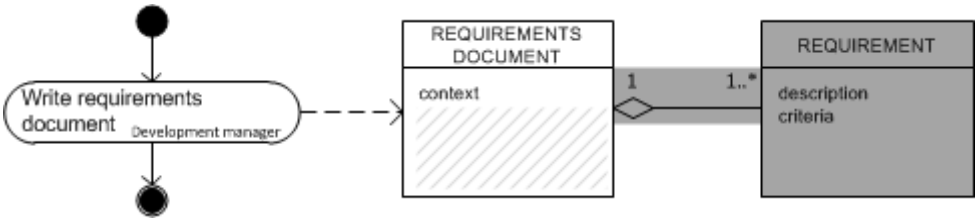


Figure A1.2. Fragment 2 of the requirement prioritization process in *Weerd et al.* (2010).

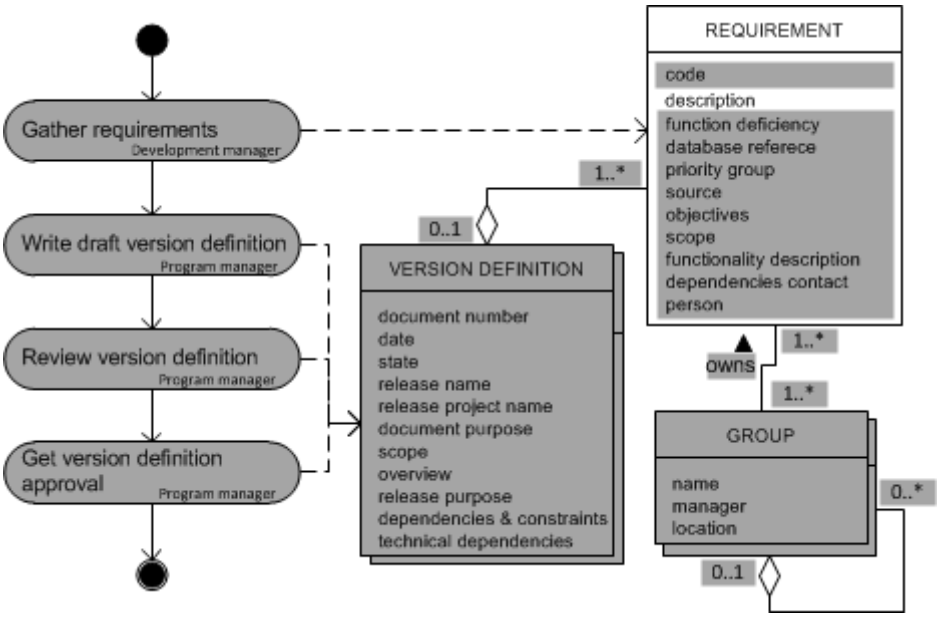


Figure A1.3. Fragment 3 of the requirement prioritization process in *Weerd et al.* (2010).

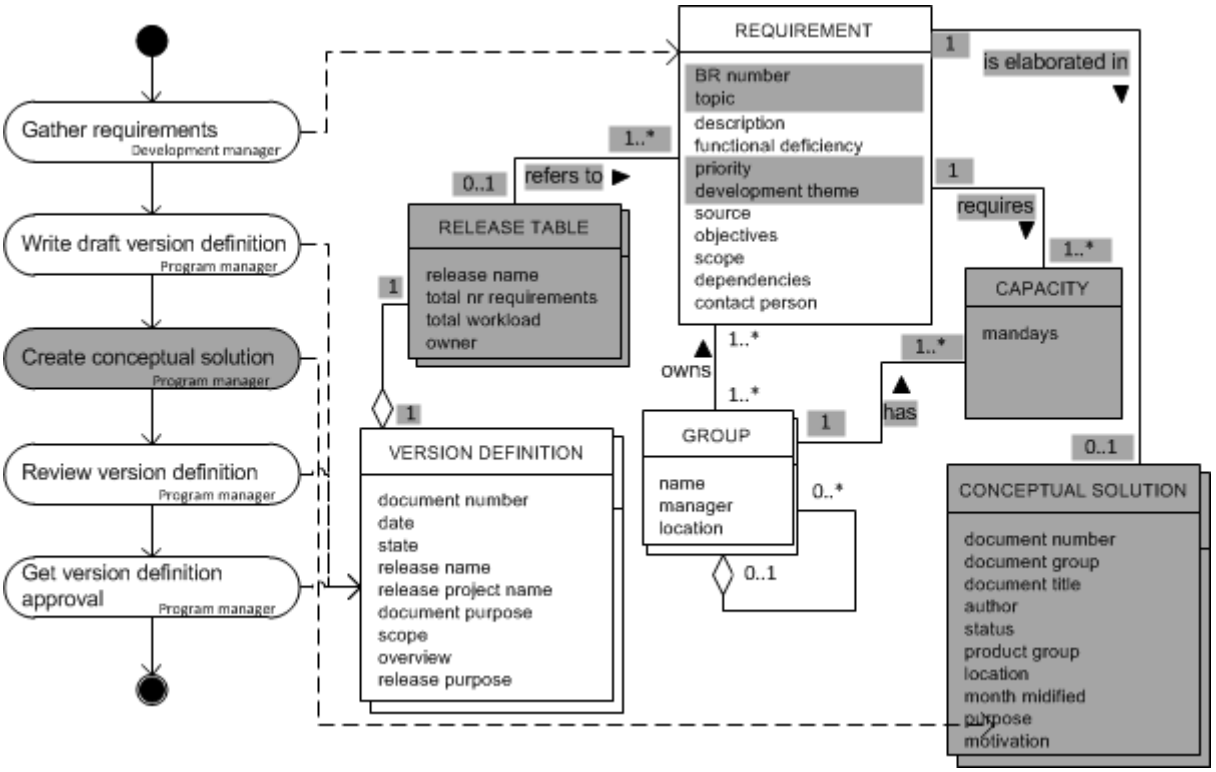


Figure A1.4. Fragment 4 of the requirement prioritization process in *Weerd et al.* (2010).

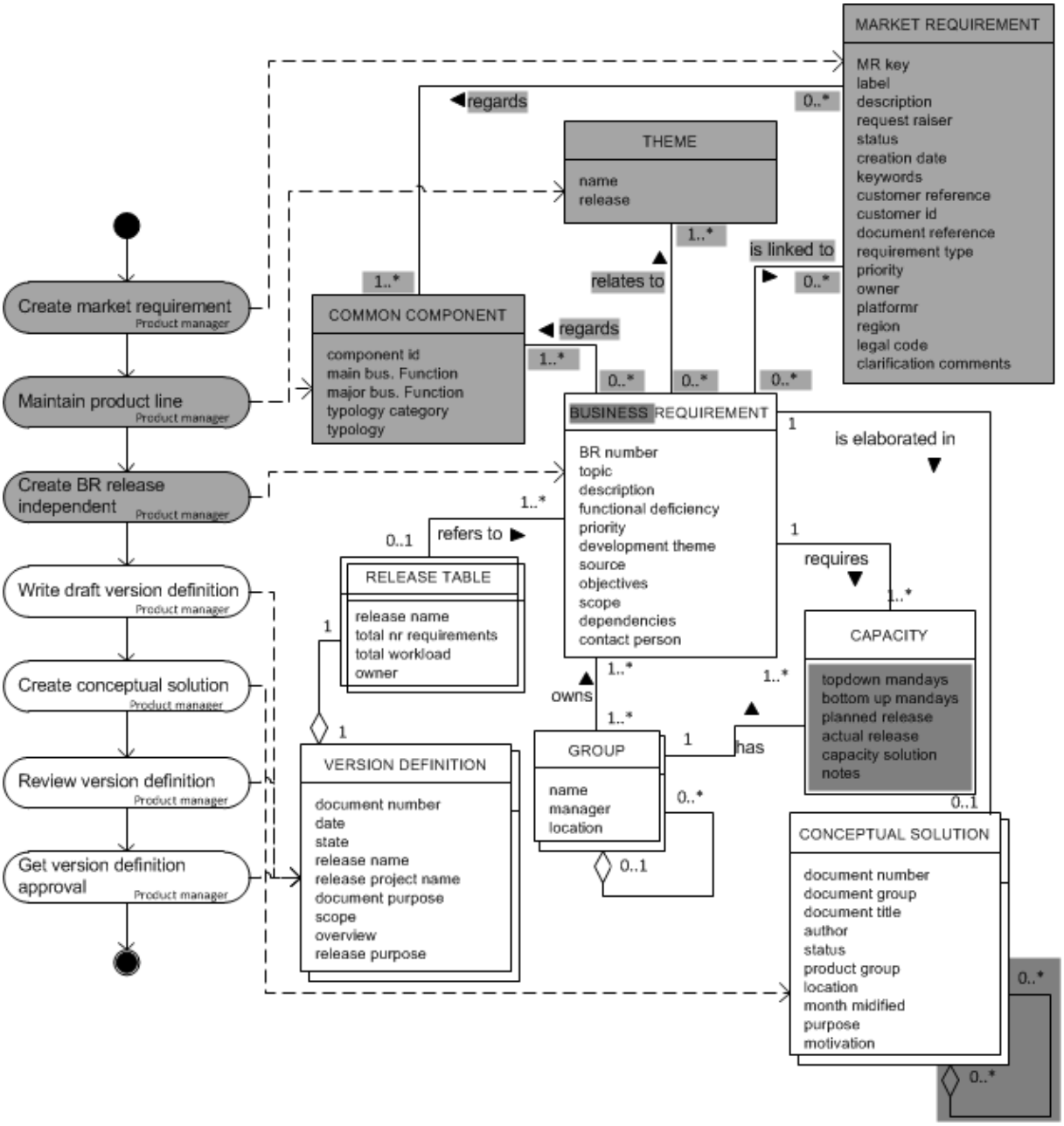


Figure A1.5. Fragment 5 of the requirement prioritization process in *Weerd et al.* (2010).

APPENDIX II

Extracting data from the fragment - UML Class diagram

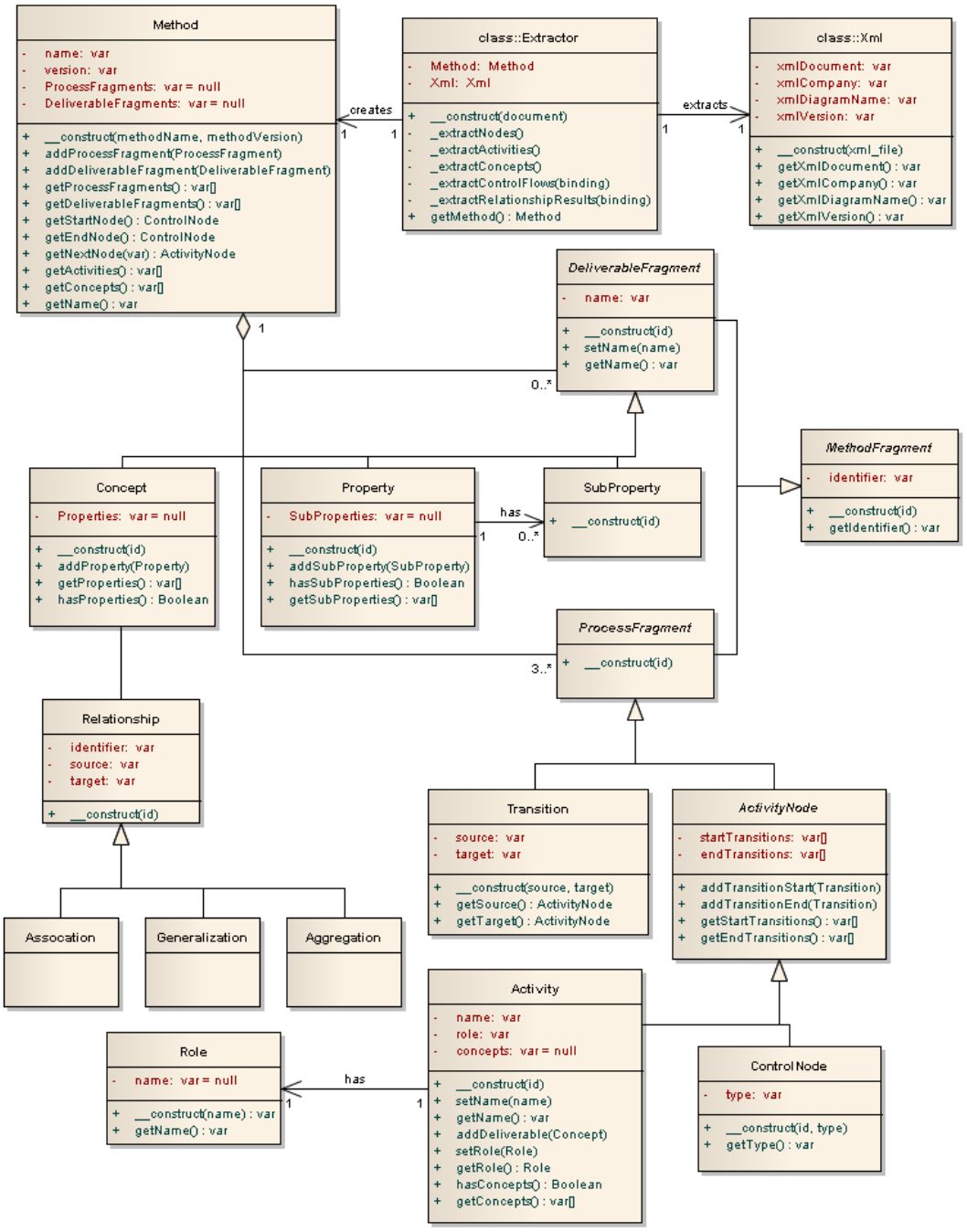


Figure A2.1. UML class diagram of the used classes for extracting data from a fragment.

APPENDIX III

Enacting a method fragment in Jira - UML class diagram

The UML class diagram for enacting a method fragment is shown below. In the UML class diagram below, only an example is provided for the Fieldscreen table in Jira. Therefore only the FieldscreenController, FieldscreenVO and FieldscreenDAO are modelled in this diagram (indicated with a yellow border).

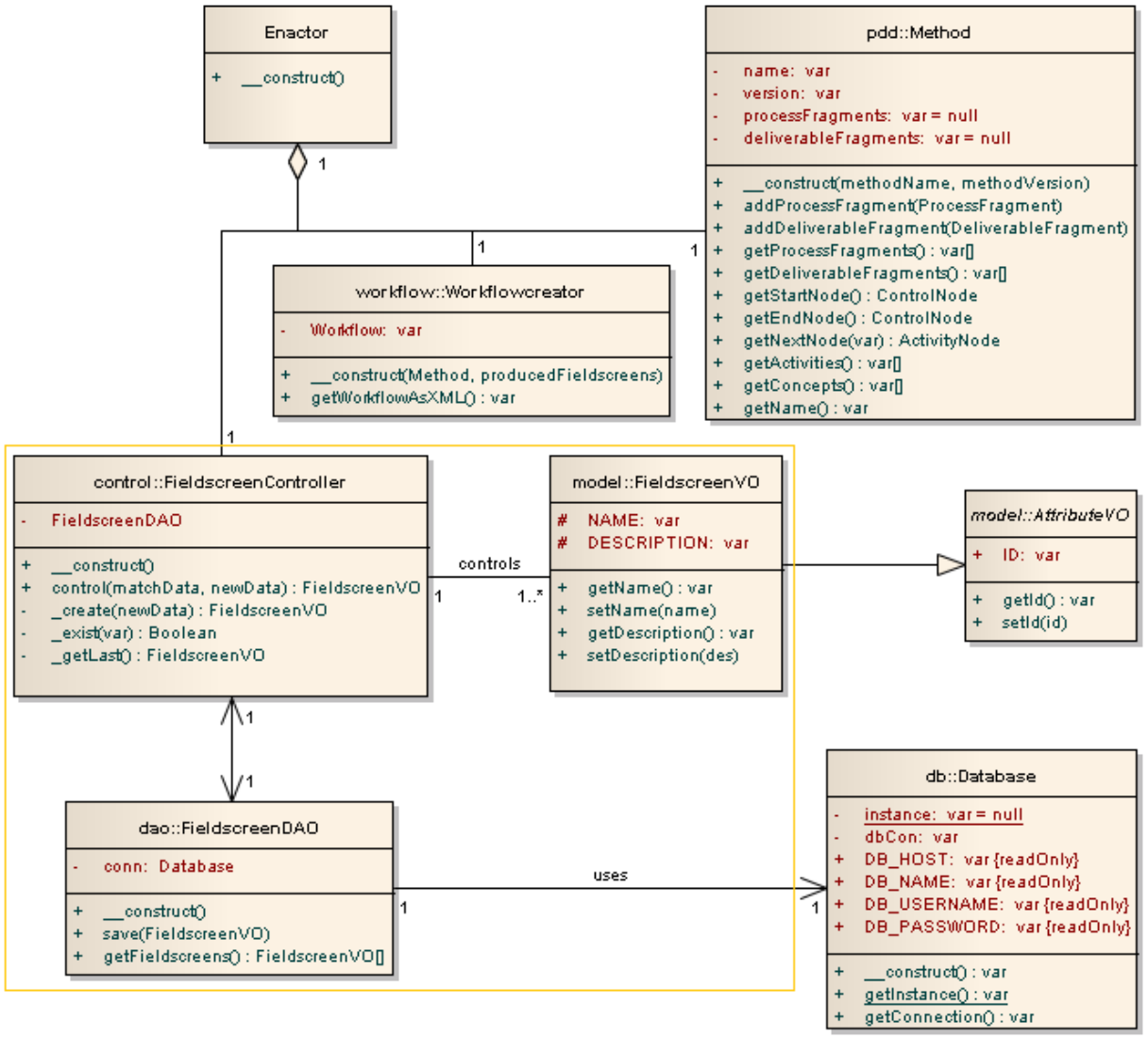


Figure A3.1. UML class diagram of the used classes for the enactment mechanism.

For each Jira table that requires data during the enactment process, we created a Controller class, a VO class and a DAO class as well. However, if we would add these classes to the UML class diagram in Figure A3.1, the diagram would become too cluttered. For that reason, Figure A3.2 depicts all the classes that are used by the mechanism. Note that these classes can be connected to the Enactor class in the same way as the Fieldscreen classes we illustrated in Figure A3.1.

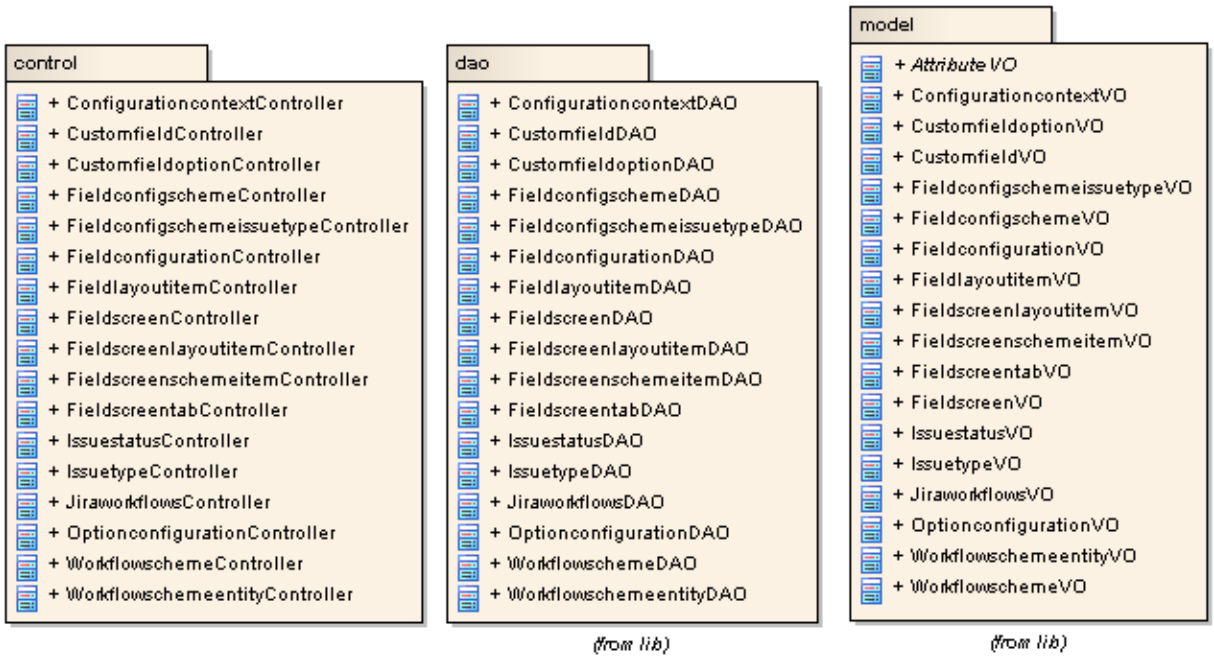


Figure A3.2. Other uses classes in the enactment mechanism.

With regard to the classes in Figure A3.2, three Controller classes (i.e. Customfield, Fieldlayoutitem and Fieldscreenlayoutitem) are equipped with an extra function to get the enactment process to work. However, the other functions and attributes in the respective classes are the same as the ones modelled in Figure A3.1.

APPENDIX IV

Interview template (Dutch)

Incremental method enactment

Interview template

Algemene details

Naam van de organisatie:

Locatie van de organisatie:

Aantal werknemers:

Datum:

Details interviewer

Naam:

Functie:

Universiteit Utrecht

Verwachte tijdsduur: 30 minuten

Details interview

Onderzoeksdoel

Het onderzoek richt zich erop hoe organisaties die software ontwikkelen omgaan met procesverandering. Vaak zijn binnen organisaties procesmodellen aanwezig die weergeven hoe men met een specifieke situatie om moet gaan. Wanneer dit proces echter evolueert op basis van nieuwe ervaringen/ideeën, dan verandert het procesmodel voortdurend op een incrementele manier. Wat echter vaak vergeten wordt is dat medewerkers zich veelal niet bewust zijn van deze veranderingen en daardoor vaak blijven hangen in oude gewoonten. De bestaande procesmodellen bieden dan geen goed beeld meer van de daadwerkelijke situatie op de werkvloer. Het is dus zaak dat medewerkers continue op de hoogte zijn van de laatste ontwikkelingen aangaande de procesmodellen.

Dit onderzoek richt zich op het dynamisch wijzigen van de ontwikkelomgeving waarin medewerkers werken. Met andere woorden, de ontwikkelomgeving waarmee medewerkers werken wijzigt op basis van de procesmodellen die worden ontwikkeld. Hiermee wordt voorkomen dat medewerkers stappen of informatie overslaan die van belang zijn in het proces. Dit onderzoek onderscheidt zich van bestaande oplossingen doordat dit onderzoek zich zowel op de proceskant als op de informatiekant richt van een procesmodel. Het onderzoek is uitgevoerd met de ontwikkelomgeving MetaEdit+ (een methode ontwikkeltool) en de project tracking software genaamd Jira (project tracking tool voor softwareontwikkeling).

Wat wordt van u verwacht?

Door middel van een aantal gerichte vragen doen wij een beroep op de huidige situatie in uw organisatie zodat vervolgonderzoek beter toegespitst kan worden op dergelijke situaties. Het onderzoek wordt geheel anoniem uitgevoerd; de informatie op de voorzijde van dit document is slechts aanwezig zodat de antwoorden op de vragen herleidbaar zijn naar de betreffende organisatie. Indien u besluit mee te werken aan het onderzoek is het wellicht handig om vooraf even stil te staan bij procesveranderingen binnen uw organisatie/afdeling. Hoe gaat uw organisatie/afdeling momenteel hiermee om bijvoorbeeld?

Wat is de opzet van het interview?

Als eerst zullen er enkele vragen gesteld worden over procesveranderingen binnen uw organisatie/afdeling. Vervolgens laten wij door middel van een video een mogelijke oplossing zien om procesverandering binnen organisaties te ondersteunen. Als laatste stellen wij u enkele vragen naar aanleiding van de video. In totaal worden er 11 open vragen en 3 gesloten vragen gesteld. Hierna is het interview afgelopen.

Interviewvragen – deel 1

Algemeen

- Herkent u de noodzaak van procesverandering binnen uw organisatie/afdeling? Zo ja, op wat voor manier?

Uw organisatie

- Hoe gaat uw organisatie/afdeling momenteel om met procesverandering?

- Zijn er processen binnen uw organisatie/afdeling die expliciet zichtbaar zijn voor u en uw medewerkers/collega's. Zo ja, kunt u er een aantal tonen?

- Indien er procesmodellen binnen uw organisatie/afdeling aanwezig zijn, bevatten deze
    a. een proceskant waarop men kan zien hoe men het proces moet uitvoeren?

    b. een informatie/data –kant waarop men kan zien wat een bepaalde activiteit moet produceren?

    c. rollen van personen die verantwoordelijk zijn voor die specifieke activiteit?

- Indien er procesmodellen binnen uw organisatie/afdeling aanwezig zijn, worden deze modellen dan tijdig bijgewerkt. Zo ja, op wat voor manier?

<<Uitleg over Process Deliverable Diagrams, MetaEdit+ en Jira>>

Interviewvragen – deel 2

Voordat u de volgende vragen beantwoordt is het belangrijk dat u uw antwoorden baseert op basis van het concept achter de aanpak. Daarom vragen wij u de zojuist getoonde aanpak niet te zien als een totaaloplossing en het ook niet te vergelijken met andere (bestaande) oplossingen.

Uzelf

- Wat is uw eerste indruk als u naar de aanpak kijkt in de demo?

- Wat vindt u van de aanpak in de demo?

- Hoe zou u medewerkers op de hoogte stellen wanneer u een dergelijke aanpak binnen uw organisatie/afdeling zou implementeren?

- Indien u aangewezen wordt om deze aanpak te implementeren binnen uw organisatie/afdeling, wat zou u dan belangrijke factoren vinden die meegenomen moeten worden tijdens de implementatie?

- Verwacht u weerstand van uw medewerkers/collega's indien u een dergelijke aanpak zou implementeren? Zo ja, waarom denkt u dat?

- Als u terugblikt op de demo, denkt u dan dat er iets belangrijks mist wat is vergeten in de demo? Zo ja, wat dan?

- Zou u een dergelijke aanpak gebruiken zoals die is weergegeven in de demo? Waarom?

Dit waren de vragen. Hartelijk bedankt voor uw tijd en moeite.

Ruimte voor opmerkingen