



MASTER THESIS

DETECTING DEPENDENCIES ACROSS PROGRAMMING LANGUAGES

THEODOROS POLYCHNIATIS
Department of Information and Computing Sciences
Faculty of Science
Utrecht University
t.polychniatis@students.uu.nl

August 6, 2012

Contents

Thesis information	IV
1 Introduction	1
1.1 Context	1
1.2 Problem description	1
1.3 Research objective	2
1.4 Research questions	3
1.5 Company profile	4
1.6 Research model	4
1.6.1 Scientific relevance	5
1.6.2 Social relevance	6
1.6.3 Research contribution	6
2 Research method	7
2.1 Research structure	9
2.2 Research phase 1: Literature study approach	10
2.3 Research phase 2: Interview approach	10
2.3.1 Interview phases	11
2.3.2 Limitations	14
2.4 Research phase 3: State of the art evaluation	14
2.5 Research phase 4, 5 & 6: Experimental setup, data analysis, method development and evaluation	14
2.5.1 Ground truth method	15
2.5.2 Selection criteria for case study	16
2.5.3 Evaluation process	17
3 Related literature	19
3.1 Concepts and definitions	19
3.1.1 What is a “dependency”?	19
3.1.2 Classification	20
3.2 State of the art	22
3.2.1 Eclipse MoDisco	23
3.2.2 PAMOMO	23
3.2.3 GenDeMoG	24
3.2.4 Moose	24
3.2.5 DSketch	24

3.2.6	Source Navigator NG	25
4	State of the art evaluation	27
4.1	Requirements elicitation	27
4.2	Rating criteria for requirements	29
4.3	Must have requirements	32
4.4	Should have requirements	34
4.5	Could have requirements	35
4.6	Prerequisites	36
4.7	Comparison matrix	37
5	Proposed method	41
5.1	General description	41
5.2	Preparation for the evaluation process	42
5.2.1	Case study	42
5.2.2	Experiment setup	43
5.3	SIG framework	43
5.4	Infrastructure	44
5.5	Pattern based approach	46
5.5.1	Database dependency type	46
5.5.2	File I/O dependency type	49
5.6	Token based approach - Ad-hoc filtering	49
5.6.1	Basic algorithm	50
5.6.2	Base case	51
5.6.3	Filtering out language reserved words	51
5.6.4	Filtering out project specific words	52
5.6.5	Tailored tokenising of XML type documents	53
5.6.6	Evaluation of token based approach with Ad-hoc filtering	55
5.7	Token based approach - Statistical filtering	55
5.7.1	Filtering out frequent words	56
5.7.2	Filtering out frequent words per language	57
5.7.3	Filtering out frequent words per file and language	58
5.7.4	Weight filtering	60
5.7.5	Composite experiment	61
5.7.6	Comparison between statistical and ad-hoc approaches	61
5.8	Ground truth data (<i>GTD</i>) amendment	63
5.8.1	Adding <i>Indirect</i> dependencies	63
5.8.2	Treating different XML based technologies as different languages	64
5.9	Filtering out frequent words per file and language category	65
5.9.1	Description	65
5.9.2	Composite experiment with amended <i>GTD</i>	65
6	Large scale case study project	69
6.1	Project description	69

Contents

6.2	Extension for another language of the project	70
6.3	Challenges	70
6.4	Evaluation method for large scale systems	71
6.5	Experiment	71
6.6	Refinement	71
6.7	Evaluation	73
7	Discussion	75
7.1	Answers to research questions	75
7.2	Validity & Reliability	77
7.3	Future research	81
8	Conclusion	83
	References	85
	Appendix	88
	Appendix A - Interview form for defining evaluation criteria	88
	Appendix B - <i>GTD</i> for small case study project	94

Thesis information

Additional information about this thesis is provided in the table below:

Thesis information

Author:

Name	Theodoros Polychniatis
Student number	3614646
E-mail address	t.polychniatis@students.uu.nl; thodpol@gmail.com

Master's Course:

MSc program	MSc in Business Informatics, Utrecht University
MSc starting date	Sep. 1, 2010
Title of thesis project	<i>Detecting dependencies across programming languages</i>
Thesis start date	Dec. 1, 2011
Thesis end date	August 6, 2012

Supervisors:

From Utrecht University:

1st supervisor	dr. Jurriaan Hage (j.hage@uu.nl)
2nd supervisor	dr. Slinger Jansen (slinger@slingerjansen.nl)

From Software Improvement Group (SIG):

1st external supervisor	prof. dr. ir. Joost Visser (j.visser@sig.eu)
2nd external supervisor	Eric Bouwers, Msc (e.bouwers@sig.eu)

Abstract

Information systems, especially large scale ones, are heterogeneous, i.e., they comprise modules, which are developed in diverse programming languages. In order to assess the complexity of such systems a method, which can detect dependencies among their modules is needed. Although there is a variety of methods that detect the dependencies within a programming language, there are not many cross-language detection methods. In this thesis the state of the art methods are reviewed, using a newly proposed evaluation method and a rating schema. The reviewed methods were not highly rated according to our schema because they focus mainly on accuracy. Therefore, a new method is proposed, which focuses on both cost-effectiveness and accuracy. The proposed algorithm is simple and generic, so that it can support a new language with minimal effort. It is also explainable to a non technical audience. In order to be able to assess the new method with the proposed evaluation approach, a tool was created and series of experiments were conducted on a small case study project. The actual dependencies in the project were extracted by manual checking, and they were used as ground truth data, with which the dependencies retrieved by the proposed method were compared. Moreover, the method was executed on a large scale information system, to evaluate the degree of generalisation of the results.

Chapter 1

Introduction

1.1 Context

Programming languages have weak and strong points, and they are used mainly in the domain they are designed for. Some languages like Java and C++ are designed for multi-purpose use (General Purpose Languages - GPLs) while others are designed for a specific domain (Domain specific Languages - DSLs). Examples of DSLs are the mark-up languages of the various wiki engines or the SQL language (Van Deursen et al., 2000). Moreover, because of the fast pace of evolution in the software development domain, modern programming languages are more suitable for new projects than languages that were popular 20 years ago. Companies try to modernise their legacy information systems (LISs), but this is a time and money consuming process which includes a lot of risk. Therefore many companies modernise only part of their LIS which work alongside with these modern systems.

Because of these reasons, companies have to maintain, expand and generally deal with software systems, which comprise components developed in more than one programming language. We call these systems *heterogeneous*.

1.2 Problem description

The modernisation, expansion and generally the modification of heterogeneous information systems is a difficult and complex task. This is the reason why many organisations hire consultancy companies with specialisation in assessment and improvement of information systems, to help them carry out and manage modernisation projects. One of the first steps in this process is the evaluation of the organisation's existing systems. The knowledge acquired from this step is crucial because the outcome of this process plays an important role in the decisions to be made by the organisation's or company's management board.

The degree of difficulty for the modification of software project is strongly and inversely

dependent upon its maintainability. According to the ISO/IEC 25010 quality standard (ISO25010, 2011), “*maintainability is the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers*”. The same standard also describes the evaluation process for the quality attributes of the software product. Although it is not necessary for every evaluation to have full compliance with the standard process, the latter provides a guideline for evaluators. In this thesis the term “*evaluator*” refers to the role of a person that conducts an evaluation of the software quality of a software system.

In many systems, especially in large scale ones, a general overview of the components of the information system makes it more comprehensible to the evaluators. Such an overview should display the dependencies of the involved components even if these components have been developed in different programming languages. Source code analysis and, in particular, static analysis in homogeneous systems is quite advanced and there is a lot of literature and tooling to support dependency detection, at least in GPLs and in popular DSLs systems. Problems arise when a system contains modules written in different languages that communicate with each other. These dependencies can take many different (syntactical) forms, can be dynamic rather than static, and may be indirect via frameworks, databases and scripts. The high number of possible combinations of languages that can exist exacerbates the problem to a greater degree.

1.3 Research objective

In this research we try to give a solution to the problem that was described in the previous paragraphs. The goal/question/metric method - GQM (Basili et al., 1994) is used to structure our objective, formulate the research questions and define the metrics to answer these questions. The objective is defined as:

The objective of this study is to improve the process of detecting dependencies across programming languages, focusing on the cost-effectiveness and the accuracy, from the viewpoint of a software evaluator, in the context of evaluating heterogeneous information systems.

The following definitions are important because they clarify the objective.

Wilde (1990) gives the following definition for “dependency”:

“From the point of view of the maintainer, there is a dependency between two components if a change to one may have an impact that will require changes to the other.”

Wilde uses the term *component* as in *software module* or in *software artefact* which is defined as “*an entity that belongs to a software system, e.g., a package, a file, a function,*

a line of code, a database query, a piece of documentation, or a test case” (Beyer and Noack, 2005).

In this context, *components* are defined as meaningful collections of software modules e.g., folders, group of folders responsible for a certain functionality, etc.

1.4 Research questions

The aforementioned problem and the objective that needs to be reached form our research question:

What is a cost-effective, yet accurate method of detecting the dependencies across software modules, in heterogeneous software systems?

And the sub-questions that arise:

SQ₁: What is a “dependency”?
SQ₂: What types of dependencies exist across languages?
SQ₃: How can we detect these dependencies?
SQ₄: How can we evaluate the state of the art?
SQ₅: How can we evaluate the accuracy of the method?
SQ₆: How can we evaluate the cost-effectiveness of the method?

The definition of “dependency” is already given in section 1.3 in order to make the document more comprehensible, thus the first sub-question is already answered. However this definition is an outcome of literature study as it is explained in section 3.1.

1.5 Company profile

In the context of this thesis, the author is working as an intern at the company Software Improvement Group - SIG¹ which presents itself as: “...a consultancy firm that provides impartial, objective, verifiable and quantitative assessments of risks related to your corporate software systems. Our analysis software allows us to measure quality, to lay bare the underlying architecture and to assess the risks related to a system.”¹



The company provides a number of services related to consultancy, such as:

- *Project risk estimation* - consultancy about the risks involved in a project plan.
- *Software risk assessment* - in-depth analysis of the technical quality of a software system to expose the risks that the company takes.
- *Software monitoring* - automated monitoring of the product quality of a software system being developed or maintained (with consultants interpreting the results).
- *Software product certification* - With a method verified by the German certification institute TÜV, SIG certifies the quality of the maintainability of a software product according to the ISO/IEC 25010 quality standard (ISO25010, 2011).
- *Sustainability scans* - for greener and more cost efficient IT.

The company employs a professional team of consultants which consists of two types of members: “*Technical consultants*” who are responsible for evaluating the quality of a software system from a client company and “*general consultants*” who receive the evaluation result from the technical consultants, translate it into business risks or objectives and advise the client company accordingly. Technical consultants have the role of *evaluators* as defined in section 1.2.

Software analysis is the main part of the services that the company provides. SIG focuses on static source code analysis because dynamic software analysis requires execution of the code and this is not always possible at SIG’s environment. Our research topic also focuses on static software analysis, hence it serves directly the company’s needs.

1.6 Research model

In this study we follow the IS research framework (see Fig. 1.1) which was proposed by Hevner et al. (2004) because it does not focus only on the creation and evaluation of an innovative IT product, but also on the impact that the research has on the environment.

As we can see from the left side of Fig. 1.1, the environment, people and organisations,

¹Software Improvement Group - <http://www.sig.eu>

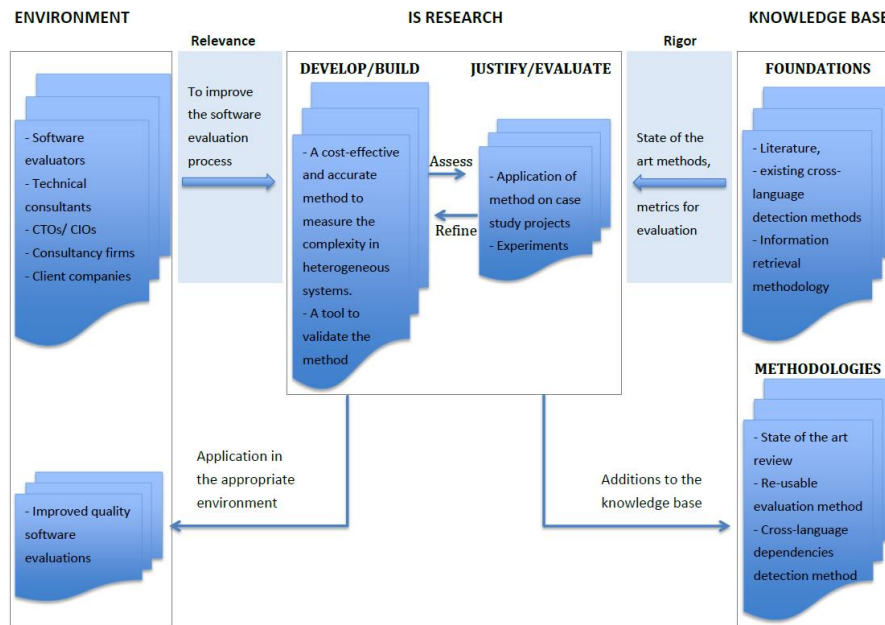


Figure 1.1: The information science research framework (Hevner et al., 2004) instantiated for this thesis.

define the problem domain which this study is inspired on. The knowledge base on the right side of the model provides the appropriate knowledge from already existing research that can be applied to our case. After some cycles of development, assessment, evaluation and refinement, an artefact is expected to be produced which is an addition to the knowledge base and is applicable to the environment. The social and scientific relevance and the contribution that is expected from this study are explained in the following paragraphs.

1.6.1 Scientific relevance

The problem of dependency finding concerns both the academic society and the software industry. Most scientific literature deals with homogeneous systems or systems written in well co-operating languages (e.g., Java with JSP). In the industrial sector there is a variety of proprietary and open-source tools that support methods for dependency finding, in particular for specific modern GPLs. However, only a few proprietary tools deal with less modern GPLs or with a small range of DSLs. The current study is relevant in the scientific domain because the success of our proposed method can fill this gap.

Moreover, the existing methods which are reviewed in section 3.2, do not define a reusable evaluation method. Validation and evaluation are conducted with interviews by experts from the case study organisations. In this study a repeatable evaluation process for the

findings of a dependency detection method is provided as well.

1.6.2 Social relevance

When software evaluators try to evaluate the quality of the information systems in an organisation, they encounter complex systems where they do not know if and to what degree the systems modules interface. To make matters worse, in many cases documentation is lacking and the expert staff responsible for the development of these systems are not working in the organisation any more. Trying to analyse the code and attain an understanding of the interconnected modules of the system is so expensive, in terms of time and effort, that the evaluators have to avoid it. Consequently the evaluators take decisions without knowledge of the dependencies, leading to failures and other unforeseen problems.

Generating an overview of this network of dependent modules, even if not 100% precise, allows the evaluators to save working hours, which they can spend on improving the rest of the modification project.

Furthermore, they will provide the organisation's management board with more data earlier in the process, to allow for better decisions and planning for the costs and the duration of the modification project. This increase in the efficiency of the evaluation process, is a common interest of the software evaluators, the organisation's CIO, CTO and generally, its management board.

1.6.3 Research contribution

We identify the following academic research contributions:

- A state of the art review and comparison.
- Patterns of dependencies across technology boundaries.
- A method for automated dependency detection to support precise enough program understanding.
- A description of a reusable evaluation process for such a method.

And the (research) contributions to the company are:

- A method for software evaluators to evaluate in a cost-effective, yet accurate way, the complexity in heterogeneous systems.
- Improved quality of consulting.

Chapter 2

Research method

In order to provide answers to the research questions, we need to decompose each research question and find the necessary research approaches through which these questions can be answered. For the sub-questions whose answers can be quantified metrics are identified, following the GQM paradigm (Basili et al., 1994).

Table 2.1 displays the research approach for every sub-question.

Table 2.1: Research approach to answer the sub-questions.

Research sub-question	Approach
<i>SQ₁: What is a dependency?</i>	-Literature study of the concepts in the field of dependency detection.
<i>SQ₂: What types of dependencies exist across languages?</i>	-Literature study of the classification of dependencies. -Interviews with experts to discover the types of dependencies experts are interested in.
<i>SQ₃: How can we detect these dependencies?</i>	-Literature study of the state of the art. -Extending or developing a method if the previous approaches are unsatisfactory. -Creating a tool in order to be able to execute the method. -Selecting case studies to execute the method.
<i>SQ₄: How can we evaluate the state of the art?</i>	-Literature study for selecting the requirements and their metrics. -Interviews with experts to validate the requirements and define their metrics.
<i>SQ₅: How can we evaluate the accuracy of the method?</i>	-Literature study for metrics if not covered by the metrics of <i>SQ₄</i> . -Evaluating the results of the developed method on experiments conducted on the case studies.
<i>SQ₆: How can we evaluate the cost-effectiveness of the method?</i>	-Experimenting by extending the (tool of the) method to handle a new language.

2.1 Research structure

The research approaches that we defined in Table 2.1, are not necessarily answered in the same order as displayed in the table, i.e., the literature study precedes all other approaches. The thesis itself is structured in phases as shown in Table 2.2.

Table 2.2: Phases of the research process

Phase	Description - Actions involved
Phase 1: Literature study	<ul style="list-style-type: none"> -Getting familiar with the research area, getting/creating definitions of concepts and terminology. -Studying classifications of dependencies. -Studying the state of the art approaches. -Defining the requirements for the main approaches. -Studying the metrics which are used for the requirements.
Phase 2: Interviews	<ul style="list-style-type: none"> -Conducting interviews with experts to validate, and if necessary, modify the requirements and define their metrics.
Phase 3: State of the art evaluation	<ul style="list-style-type: none"> -Evaluating the state of the art and making the decision of extending an already existing method or developing a new method in order to achieve our objective. As described in section 4.7, the decision is to develop a new method/algorithm because the state of the art does not meet the requirements.
Phase 4: Experimental setup and data analysis	<p>This phase involves:</p> <ul style="list-style-type: none"> -defining the implementation technique, -selecting case study projects where the method should be applied, -defining the evaluation approach of the method.
Phase 5: Method development and evaluation	<p>This phase involves an iteration of:</p> <ul style="list-style-type: none"> -designing the method/algorithm, -implementing the algorithm, -testing the algorithm for defects, -experimenting with different parametrisation, -evaluating results.
Phase 6: Final evaluation	<p>The final evaluation involves:</p> <ul style="list-style-type: none"> -A large scale project case study -An extensibility experiment.
Phase 7: Conclusion	<p>The final phase involves discussion of the results, identification of future research opportunities and conclusions.</p>

2.2 Research phase 1: Literature study approach

The first phase of the research is performed by conducting a literature study. Books, journal articles, conference papers, published literature reviews and international standards are the sources of information for this research. These sources are accessed through free web-based search engines, tailored for academic purposes, such as Scirus¹, Google Scholar², Citeseer³ and Mendeley⁴.

Following the guidelines in (Levy and Ellis, 2006), the literature study started by searching for publications containing keywords and phrases related to our topic. As Levy and Ellis (2006) suggest, good quality journals were selected using the ranking list of journals published by the Association for Information Systems⁵. Therefore, published material in ACM⁶, IEEE⁷ journals and MIS Quarterly⁸ were studied first. Although the rigour of leading journals is higher than that found in conference proceedings, as Webster and Watson (2002) suggest, the publications from high ranking conferences were also examined.

To expand the study material, backward and forward search (Webster and Watson, 2002) was conducted, i.e., the gathering of articles following the references and the citations from the set of articles which are already examined.

For each article we studied, the set of concepts and methods that are described were extracted. These findings were examined and evaluated in terms of the applicability that they may have in our research.

The end of the literature study phase came to an end, when the same references were found and no new relevant concepts were discovered (Levy and Ellis, 2006).

2.3 Research phase 2: Interview approach

In order to evaluate the state of the art of the methods which can detect cross language dependencies, requirements (RQs) for such methods need to be defined. These RQs are formed by conducting semi-structured interviews with general and technical consultants from SIG. The RQs of the method are directly translated into RQs for the software which implements the method. Although RQs are usually separated into functional, quality RQs and constraints, during the elicitation phase, no separation is made, because we

¹<http://www.scirus.com/>

²<http://scholar.google.nl/>

³<http://citeseer.ist.psu.edu/>

⁴<http://www.mendeley.com/>

⁵<http://ais.affiniscap.com/displaycommon.cfm?an=1&subarticlenbr=432>

⁶<http://www.acm.org/>

⁷<http://www.ieee.org/>

⁸<http://www.misq.org/>

want to capture the most important RQs regardless of their category. The interview approach is described in the next paragraphs.

2.3.1 Interview phases

Preparation

For the preparation of the interview the following key points were considered:

1. *Goal definition:*
 - To define and prioritise the requirements for a cross language dependency analysis method.
 - To collect other interesting information for this research which the interviewees may provide.
2. *Selection of interviewees:*

Six consultants were selected from SIG. The interviewees have at least two years of experience in evaluation. Three of them are technical consultants, whose main role is to evaluate software, and the other three are general consultants, who, together with the technical consultants, identify risks. These two types of consultants were needed for our interviews, in order to capture the rationale from both aspects of the evaluation process.
3. *Type of the interview:*

For the first goal a structured interview (Kvale and Brinkmann, 2009) is appropriate because the same questions with the same wording and in the same sequence are asked to all the respondents. For the second goal, where any additional information is captured an unstructured explorative approach is more suitable.
4. *Structure of the interview:*

The interview starts with an introductory session. Then it is divided into four phases. The first three concern the first goal and are of a structured type and the fourth phase is explorative and concerns the second goal of our interview.
5. *Duration:*

Due to resource constraints, the duration of the whole process has to be less than 35 minutes unless the interviewee wants to provide us with more information.

The interview form that was used is available in the Appendix A.

Introductory session

The aim of the introductory session is to communicate the objective of this thesis and the goal of the interview to the interviewee. It is important for the interviewee to understand the purpose of a method that detects dependencies across language boundaries. This session lasts for five minutes.

Phase 1 - Requirements discovery

After we confirm that the interviewee has a good understanding of the objective, he/she is asked to name the top five RQs for such a method. We note down each RQ and we make sure that we have understood it. It is not necessary for the interviewee to provide all five RQs, because this might extend the scheduled duration of the phase which should be ten minutes.

Phase 2 - Requirements prioritisation

In this phase the interviewee is given a set of RQs (listed in Table 2.3), which are believed to be important for this method. This set of RQs was created by:

- brainstorming, using the author’s professional experience in developing multi-system enterprise applications,
- studying the literature,
- unstructured interview with one experienced technical consultant and the quality manager from the same company.

Table 2.3: Requirement set presented to the interviewees

Requirements	Description
Explainability	The meaning of the dependency findings should be able to be understood by the software evaluator and the customer.
Accuracy	The percentage of False Positives and False Negatives* should be as low as possible.
Tool integration	The method should be able to be implemented as part of an existing tool.
Visualisation	The results should be displayed in a visual way.
Filtering	The results can be filtered (e.g., according to keywords or dependency types).
Initial range of languages	The method initially should support a set of different programming languages.
Extensibility	The method should be extendable so as to include an unsupported language in a short time-frame.
Efficiency	It should be fast enough so that it can be executed on large-scale systems in a reasonable time-frame.
Exportability	The results should be exportable to open format(s).

**False Positives* are dependencies that are found by the method but are not actually true dependencies. *False Negatives* are actual dependencies that the examined system has, but the method does not find.

The set of the RQs is given to the interviewees to discover whether extensibility and accuracy, which are the initial focus of our study, are important for them as well. To prevent biasing the interviewees, our set is not presented in the beginning. For the same reason the order of the requirements for each interviewee is randomised. The goal of this

phase is to get the interviewees opinion about the priority of each RQ.

The MoSCoW technique (Knudsen, 2004) was selected for the prioritisation because the evaluators are usually already familiar with it and if not, it is very simple to understand it. It is also very fast in comparison with other techniques like Cost-Value Approach (which uses Analytical Hierarchy Process) or cumulative voting (Ahl, 2005), (Berander and Andrews, 2005).

In short, RQs are divided into the four following categories:

1. *Must have* - Critical RQs, if they are missing, the project is considered to be a failure.
2. *Should have* - High priority RQs. The project however can be delivered without them.
3. *Could have* - Desirable, but not necessary RQs.
4. *Won't have* - RQs which will not be included in the project.

While the interviewee reads the RQ set, we fill in the MoSCoW table with the requirements that he/she selected in the first phase. The interviewee's RQs are placed before the predefined set, to reduce the possible bias that could be introduced by placing our RQs first.

Then the MoSCoW technique is introduced to the interviewee, while the prioritisation table is presented. The interviewee has the opportunity to add more RQs if he/she did not think of them during the first phase. The interviewee is informed that the time-frame of the development of the method is 4.5 months.

The whole duration of second phase should not exceed 15 minutes.

Phase 3 - RQs quantification

In this phase we ask for the interviewee's opinion about the value, or a range of values for each RQ, e.g., for the extensibility of the method, the value could be in a number of man-days. This information makes our understanding of the RQs more substantial because it quantifies them by introducing a measuring scale. Five minutes are enough for this phase. With this phase, the structured part of the interview ends.

Phase 4 - Additional information

In the fourth phase we try to capture any additional information from the interviewees that could help our research. They are all experienced consultants and they can possibly have a particular idea, event, or any other piece of knowledge to share, i.e projects from the company that would be interesting case studies, or ideas for the implementation of the method. One of the questions here is what type of dependencies the consultants find interesting to be extracted by the method. Because of the explorative nature of this part, we decided to keep it unstructured.

2.3.2 Limitations

A fact that might be considered as a weakness of our method, is that our predefined set of RQs was presented, instead of organising workshops with the consultants in order to discover and prioritise the RQs.

Another limitation was that because of time constraints, the number of the interviewees was limited and selected among the employees of one company. This threatens the generalisability of the results because the interviewees have acquired their working experience as consultants mostly at the same company, which means that they share a common way of thinking.

However we feel that these problems are not counter our research because the objective of our research, targets in a great degree the exact domain of the company.

2.4 Research phase 3: State of the art evaluation

In the third research phase the information from the literature and the expert interviews is used to create a rating schema with which the state of the art methods for detecting dependencies is evaluated.

For each requirement a metric is defined using a five level Likert-type scale (Arnold et al., 1967). Every level of that scale is defined by a statement. The statements were created by using the experts answers to the interviews, and by brainstorming when the answers were not given in detail.

Once the rating schema was created, we tried to deploy the tools that are provided by the corresponding state of the art methods. After the successful deployment of the tool, the methods were tested and evaluated against the rating schema. As it is mentioned in section 4.7, only three out of six methods were deployable and usable. In order to evaluate the unusable methods, a search for publications that refer to these methods in the literature was conducted, according to our approach in section 2.2.

2.5 Research phase 4, 5 & 6: Experimental setup, data analysis, method development and evaluation

With the completion of the third phase, as is mentioned in section 4.7, it was decided not to base our research on one of the state of the art methods, but to introduce a new method. This phase describes the research approach for the creation of the new method.

The experimental setup is an important step before we proceed with the development

of the method, because it allows us to create a framework to methodically evaluate the data that is acquired with the execution of the proposed method (*PM*). Our goal is to describe a reproducible and unbiased evaluation process.

The evaluation of our method requires:

- A second method for finding cross language dependencies with accurate results, or with a known error rate, so that its results can be compared with the results of our method. We call the second method the ground truth method (*GTM*), since it gives us ground truth data (*GTD*) about the dependencies that exist in the software system.
- A case study project on which dependency extraction methods can be executed.

2.5.1 Ground truth method

Because the *PM* extracts dependencies which belong to a set of different dependency types, the *GTM* should be able to discover the dependencies for these patterns. The state of the art methods which are reviewed in chapter 3 are candidates for *GTM*s. However we deem them unsuitable for our purpose because of the following reasons:

- The most important reason is that for the methods that were applicable, there was no evidence for their accuracy⁹, hence their error rate is unknown. What could be done, is compare the results with our method and have more confidence for the results that both methods have in common. However, manual checks for distinctions would be necessary.
- Their maturity is low⁹, meaning that their deployment, configuration and usage is difficult and there is no proper documentation to overcome these difficulties. Therefore, a lot of time is needed to understand and use these methods as *GTM*s.
- The reviewed methods support only a small set of languages, or a specific type of dependencies⁹. This means that these methods need to be extended in order to use them for a comparison with our method's results. Although that would be ideal, the time-frame of this thesis does not allow for this.

Manual checking of the source code modules was selected to be the *GTM*. By “manual checking” we mean that the subject studies a set of source files at hand and detects dependencies according to his/her experience. The detailed process is explained in the next section.

The disadvantages of manual checking are:

- It is not automatic, thus it is time consuming.
- Because the evaluator searches for dependency patterns which he, himself developed, manual checking is biased to a greater degree. The *GTM* could be performed by another software evaluator to eliminate this bias. This requires time, which the

⁹The reader can refer to section 4.7 where the state of the art methods are evaluated.

software evaluators from the company do not have.

The advantages are :

- Traditionally, manual checking is well accepted by software evaluators as a cross-language dependency detection method, hence it is admissible to use it in our case.
- It can show the accuracy of the results in a common way for any language combination or pattern used. This increases the explainability of the results and allows for the comparison of accuracy among languages and patterns.

2.5.2 Selection criteria for case study

An important factor in the evaluation process is the selection of a case study project. First of all, a multilingual project is selected in which we know, from its description, or from its functionality, that there are dependencies among the modules of the system. Either recent projects from the company's client portfolio can be selected, or open-source projects. Each choice has advantages and disadvantages:

- Selecting an open-source project increases the transparency of the results. The method can be executed repeatedly by scientists and the expected results should be the same. Therefore the experiment can be easily reproduced.
- On the other hand, by selecting a project from the company archive the guidance of the software evaluator who was assigned to that project is provided. The information from the evaluator can reveal more cross language dependencies than we might expect. The drawback is that the source code of the company projects is regarded as confidential information and is not publicly available for future experiments.

To have a reproducible experiment, an open-source project is selected. In order to get some more information about where the dependencies can be found, the software evaluators from SIG who evaluated systems with similar technologies were consulted.

A summary of the dilemmas and the decisions taken is given in Table 2.4.

Since dependencies are detected manually, the case study project ideally has technologies that we are familiar with, so that the manual extracting of dependencies has high accuracy. The size of the project is relatively small for two reasons:

- The manual check for dependencies can be done in a limited time-frame.
- The probability of having more accurate results from the manual check increases. In a small project, all the dependencies can be manually extracted, instead of taking a sample of its files and generalizing the results.

Table 2.4: Evaluation process decisions

Decision point	Candidate solution	Pros/Cons	Selected approach
Ground truth method	reviewed methods	+ automation + low bias – specific pattern or languages – low maturity, unusable – no accuracy evidence	
	manual checking	+ common way of results interpretation + de-facto method in the industry – not automatic, time consuming – more biased	X
Case study	company project	+ evaluator’s expertise available – confidentiality of source code – not reproducible experiment	
	open-source project	+ transparency + reproducible experiment	X

2.5.3 Evaluation process

During the evaluation process, the proposed method (*PM*) is executed and evaluated against the *Must have* criteria which are discussed in section 4.2. Our method is created through an iterative process of brainstorming for algorithms, implementation, testing, evaluating and refinement (see chapter 5).

For every iteration the evaluation process involves the following steps:

1. Execute the method on the case study project.
2. Retrieve the results.
3. Compare the results with the ones retrieved with the *GTM*.
4. Calculate the accuracy metrics which were discussed in section 4.3.
5. Examine the False positives (FP) and the False Negatives (FN) to improve the algorithm.
6. Judge the algorithm in terms of explainability and extensibility.

In order to test the applicability of the method on to other projects, and be able generalise the configuration and the results that the method has in the small project, the method was executed and evaluated on a large scale project. More information is given in chapter 6.

Chapter 2. Research method

Chapter 3

Related literature

According to the research structure which was presented in section 2, the first phase of our research involves studying the literature in order to answer parts of the research sub-questions. Our goal is to:

- become acquainted with the research domain and to find definitions of related concepts and terminology,
- learn about classifications of dependencies,
- discover the state of the art methods for detecting cross-language dependencies.

3.1 Concepts and definitions

3.1.1 What is a “dependency”?

As the main objective of this thesis is to detect cross-language dependencies, we need to define the term “dependency”. This definition directly affects the way in which dependencies are extracted from the source code and the way of validating them. Although we searched the literature for a term, no standardised or commonly used definition was found. There are abstract definitions, such as the one proposed by Nagappan and Ball (2007) where a software dependency is defined as *“a relationship between two pieces of code”*.

Spanoudakis (2005) defines dependency with more detail as *“a relationship between two software entities $e1$ and $e2$, where the existence of $e1$ relies on the existence of $e2$, or changes in $e2$ have to be reflected in $e1$.”*

A similar definition, which also introduces the concept of functionality, is given by de Souza (2005): *“A dependency between software modules is said to exist when one module relies on another to perform its operations or when changes to the latter must be reflected on the former”*

The closest definition to the one we seek is given by Wilde (1990) as:

“From the point of view of the maintainer, there is a dependency between two components

if a change to one may have an impact that will require changes to the other.”

Wilde’s definition was chosen for two reasons: First, because he specifies the viewpoint (maintainer), which happens to be the viewpoint for our research. Second, because it is detailed enough for our scope and at the same time, it also allow us to cover types of dependencies where a change in one module, may require changes to another module, but not necessarily.

Note that Wilde uses the term *software component* in place of *software module* as defined in section 1.3.

3.1.2 Classification

The extraction of a dependency between two modules, pre-requires, way(s) to detect it. In order to detect the dependency, we need to understand how two modules are related. Therefore, we explore the literature to acquire more knowledge about how the different dependencies can be classified. This will also make the definition, and consequently our research more tangible.

There are different ways in which dependencies can be classified. In the approach followed by Bagchi et al. (2001), tasks or transactions in a software system are associated with the software and hardware resources being used for the specific tasks. The sequence of the resources is used to create a dependency graph and the strength of the relationship is categorised in four levels: absent, weak, medium, or strong. This categorisation suits Bagchi et al. because they focus on dynamic dependency analysis where dependencies are found during the run time of the software. This type of classification is not applicable in our case because of our focus on static analysis of the source code.

Wilde (1990), besides the definition of dependencies, introduces the following classification of dependencies:

- Data item dependencies - regarding variables, data structures, and records.
- Data type dependencies - regarding relationships between types/structures that a programmer can define within a programming language and their properties.
- Subprogram dependencies - regarding calling other programs within a program.
- Source file dependencies - inclusion of header files, imports etc.
- Source location relationships - the relationship that exists because the relative location, the packaging and the organisation of files should reflect a certain functionality grouping.
- Additional classes of dependencies - other language specific dependencies.

From Wilde’s classification, we can see that the dependency types describe dependencies where the changes in one module have a rather direct impact on the other, e.g., changing a variable, a name or a sub-program, will have a direct impact on the module(s) using it.

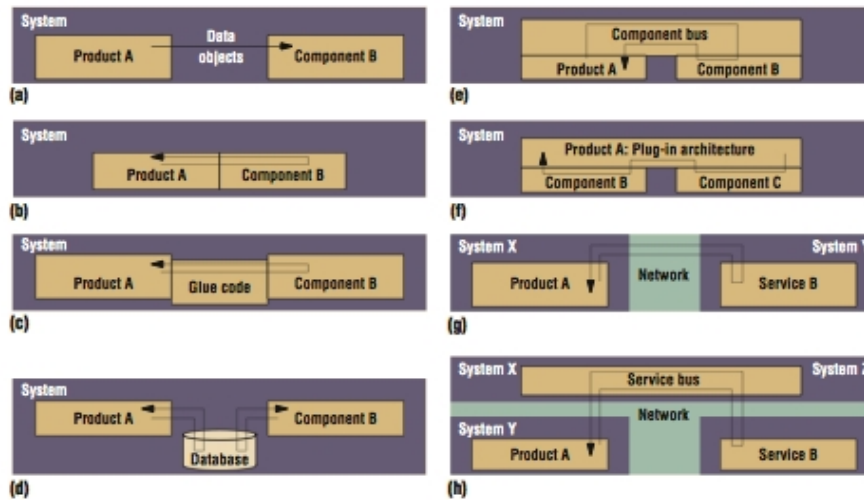


Figure 3.1: Graphical representation of SW extension mechanisms (Jansen et al., 2008)

A more architectural viewpoint of classification, which is introduced by Jansen et al. (2008), is displayed in Fig. 3.1. It depicts a graphical representation of software extension mechanisms, which implies software dependencies as well. The type of extensions, hence, the type of dependencies corresponding to the diagrams in the image are listed below:

- (a) direct data interchange among components
- (b) component reuse through libraries, e.g., method calls, dynamic link libraries, jars
- (c) glue code, when one component calls another through a wrapping interface
- (d) shared data objects, when components share the same datasource
- (e) component bus, when components are invoked as a service, e.g., Corba
- (f) plug-in architecture, e.g., Eclipse, Joomla plug-in mechanisms
- (g) service oriented architecture with pre-established conventions, e.g., SOAP calls, translation services
- (h) enterprise service bus, namely dynamic, automatic service invocation probably with service level agreements

As Jansen et al. mention in the same paper and as can be deduced by the arrows in the figures, the above interactions among components (a), (b), (g) are “direct” and the rest are “indirect” component invocations.

For this thesis this classification is modified to include the following three categories:

- *Direct* - strong dependencies among components/modules which have a change impact on each other without (necessarily) an intermediate module through which the dependency arises. These dependencies would probably cause compilation errors. e.g., program calls, file inclusions.
- *Framework* - weaker type dependencies which would probably cause runtime errors.

e.g., injection of objects/properties at runtime through application frameworks or servers like spring framework dependency injection.

- *Indirect* - even weaker dependencies, which may not cause errors at runtime but which have a high probability that will cause errors in functionality. e.g., a change in a table property in a database may affect the form in a JSP page displaying the properties of the corresponding table entity, even though there are abstraction layers between the data layer and the presentation layer.

In this thesis we focus the analysis on the dependency types that are displayed in Table 3.1 because they cover the cases of dependencies which we want to extract.

Table 3.1: Dependency types

Dependency types	Description	Possible granularity
Database access	SQL based or other types DB	Schema, Table, Column
File access	File I/O	File, class, method
Program calling	Executable programs calls	
Method calls	Direct method calls through interconnecting libraries	
Configuration files	Property files, XMLs	
Network based	Sockets, WebServices, Message Queues	

Although each type can be extracted at a different level of granularity, the focus of the dependency detection is only on the file level for the following reason: if a finer level of granularity is used, such as method level or block level, then the method will depend more on the elements of the structure that is provided by the language at hand. This means that the specific language vocabulary and syntax have to be considered. In a file level, the knowledge of the structure of the language may not be necessary. File level is also enough for our context of getting a first overview of a system's dependencies.

3.2 State of the art

In this section we present the main findings from our literature study are presented. This study does not include commercial methods/applications because: firstly, we could not have access to them to evaluate them, and secondly they are not open source, hence, we cannot learn about the algorithms they use. It is noted here that the evaluation of the methods is valid for the time that this thesis was written.

The criteria used to evaluate the state of the art are actually requirements for a dependency detection method, and are formed by the literature study and interviews with experts. Therefore they are discussed after section 4.1.

3.2.1 Eclipse MoDisco

The Eclipse MoDisco project¹ is an initiative that aims at providing a framework in the context of Model-Driven Reverse Engineering, to support the extraction and exploitation of models in legacy applications. Software models are considered as representations of software modules with a level of abstraction, that depends on the purpose of the modelling process. It considers as inputs all types of modules like source code, databases, configuration files etc. and provides the capabilities for generating their corresponding models and their handling (dependency finding, analysis, metrics, documentation, visualisation). The framework achieves that by using an extensible set of “model discoverers” that comply to the Knowledge Discovery Meta-model (KDM) (Bruneliere et al., 2010). The process involves three basic steps:

- the creation of a language specific model
- the creation of a language independent meta-model, and
- the enrichment of the language independent meta-model with additional information such as data and control flow.

The AtlanMod MegaModel Management - AM3² (Jouault et al., 2010) is a component which is responsible for managing the meta-models of each software module and the AtlanMod Model Weaver - AMW³ automatically creates links among the elements of these meta-models. Currently it provides advanced support for Java and Java related XML technologies, as well as C#, but for other GPLs and DSLs, discoverers need to be implemented. MoDisco provides the infrastructure for dependency analysis, but the patterns for weaving models still need to be written by the analyst.

3.2.2 PAMOMO

In Guerra et al. (2010) the authors use Ecore⁴ models to represent software modules and on them they apply pattern matching algorithms in order to define traces and constraints between these models. They created an eclipse plugin (Guerra and de Lara, 2010), PAMOMO, which they provide along with a simple, made-up case study. The tool provides support for defining the patterns, but the process of creating the Ecore models is not automated and there is little documentation about it.

¹The Eclipse-MDT MoDisco project: <http://eclipse.org/MoDisco>

²The Eclipse AM3 project: <http://wiki.eclipse.org/AM3>

³The Eclipse AMW project: <http://www.eclipse.org/gmt/amw>

⁴Eclipse Ecore package: <http://download.eclipse.org/modeling/emf/emf/javadoc/2.7.0/org/eclipse/emf/ecore/package-summary.html>

3.2.3 GenDeMoG

GenDeMoG (Pfeiffer and Wasowski, 2011) is one of the few dependency finding tools, based on Eclipse EMF⁵ technologies as well, which has been executed on a real, large scale heterogeneous system. Their case study included an application suite which is build in Java and many other modern XML based DSLs. The tool follows these steps:

- A Component Descriptor Model - CDM has to be defined by the user. In this descriptor the system's components are defined and initialised with properties like visibility, type, location etc. In the same file the dependency patterns in the form of queries are written, using the EMF expression language.
- These queries are then compiled into search programs.
- A graph generator transforms their output into a dependency graph for the system.

The tool can work with languages based on the Ecore meta-meta-model. Pfeiffer and Wasowski (2011) state that XSD based languages can also be automatically transformed into Ecore based ones. This is required because, if we want to link the modules which are instances of meta-models (language types), the meta-models must all belong to the same meta-meta-model (M3 level).

3.2.4 Moose

Moose (Ducasse et al., 2009) is another framework for software analysis of homogeneous and heterogeneous systems. It is written in Smalltalk and the analysis results can be displayed visually with graphs. It uses the same principle as before, namely the use of the so-called FM3 language independent meta-meta-model. According to the authors, an FM3 adhering meta-model can be created from object oriented languages, and they provide an API for that. It uses a language (MSE) similar to XML for importing new languages and transforming them into the FM3 format. PetiParser is a parsing framework provided by the same platform which enables the (manual) construction of custom parsers for new languages.

3.2.5 DSketch

DSketch (Cossette and Walker, 2010) or GrammarSketch, is the first approach found that does not use meta-modelling techniques to find dependencies. It is based on regular expression pattern matching, but provides the developer/analyst with a language to write these patterns in a much easier and meaningful way. Currently the tool is provided as an eclipse plugin and works with Java, SQL, XML. Cossette and Walker (2010) state that it is extensible without providing an API. In the two case studies that were presented in (Cossette and Walker, 2010) the developers and evaluators of the method mentioned that

⁵The Eclipse EMF project: <http://www.eclipse.org/modeling/emf/?project=emf>

the tool was relatively accurate but they would not trust it without a manual validation of the findings.

3.2.6 Source Navigator NG

Source Navigator⁶ (SN) is a source code analysis tool intended to be used in large scale systems. It can reveal the relationships among classes or their members as well as build method call trees. Language specific extractors are used to extract facts from the code which are stored along with location and scoping information into a database, for later processing. Then with cross-referencing of the facts, the relationships among them are created. Depending on the user's task (e.g., class hierarchy, call tree, include graphs), the data from the database are analysed and presented using a graphical interface written in Tcl/Tk. SN supports more than 11 languages (e.g., Java, C/C++, Python, PHP, Fortran, COBOL, Tcl/Tk etc.) which are bundled with the program but it can be extended with more languages through the provided source development toolkit (SDK).

⁶<http://sourcnav.berlios.de/>

Chapter 3. Related literature

Chapter 4

State of the art evaluation

In this chapter the state of the art methods for detecting cross language dependencies which were presented in section 3.2 are reviewed. For that, a set of requirements is created which the method should ideally have. In addition, rating criteria for each requirement is given, with which the methods are compared. After the explanation of the requirements, a comparison matrix is presented, giving information about the score of each method for each requirement.

4.1 Requirements elicitation

The requirements for a dependency detection method are formed by conducting semi-structured interviews with general and technical consultants from SIG. The interview method is explained in 2.3 and the results from the interviews are given in the next paragraphs.

The first goal of the interview sessions is to define and rank the requirements the consultants believe are important for dependency detection method. For that, the results of the second phase of the interview are compiled.

As we do not want to reveal the names of the participants, the names are substituted with variables. GC1, GC2, GC3 are the names of the three general consultants and TC1, TC2, TC3 the names of the three technical consultants.

In the first phase of the interview, participants added RQs that already existed in the given set of the second phase. Only two requirements were added by two different participants:

- "Universality" - the method should detect dependencies with the same way within a single language and across languages as well.
- "Language agnosticity" - The method should be able to detect dependencies without language specific information.

These two requirements are added to the predefined set of RQs. Then the priority of each

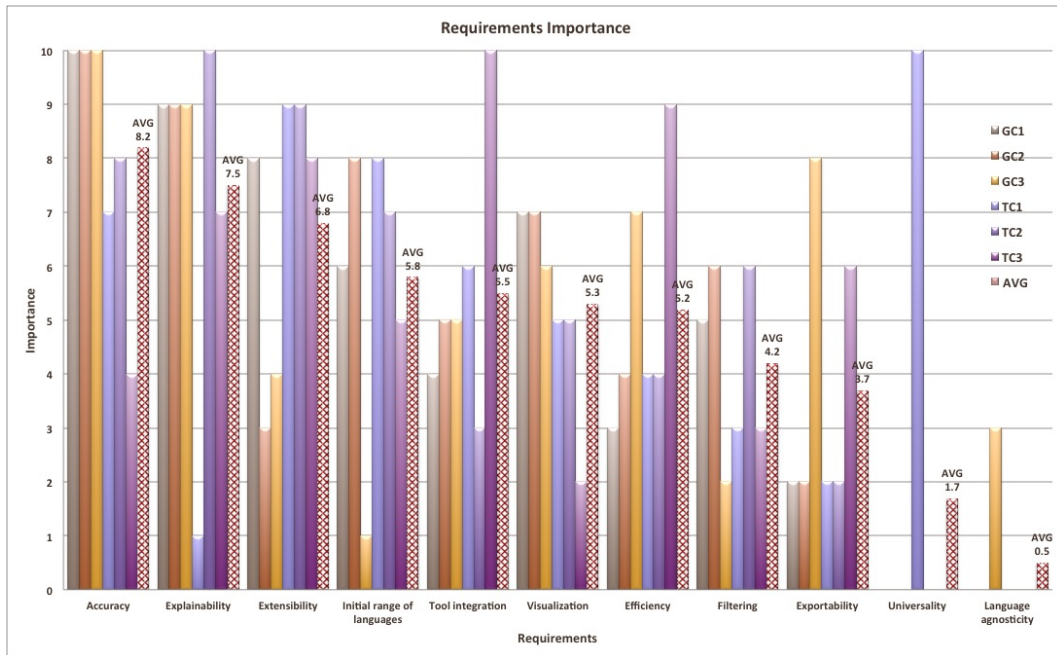


Figure 4.1: For each requirement, the answers for all 6 interviewees are shown in an 11 point scale (0 = least important, 10 = most important). The seventh bar shows the average. Interviewees were general consultants (GC) and technical consultants (TC).

RQ is calculated by the average of each RQ in the new set. A new variable is introduced in the analysis, namely "importance" which is the complement of the priority and its scale is from zero to ten (The number of the RQs is eleven). The score of the importance of each RQ by each participant, and their average is depicted in Figure 4.1:

From the results we can see that the hypothesis that accuracy and extensibility are important, is confirmed, as they are ranked in the first and third position. An interesting fact is that explainability is significant for the consultants as well. During the interviews most consultants shared the opinion that the method should provide results that can be easily understood by their clients. The metrics used by the method should be simple, so that the clients can be convinced that the dependency analysis which is applied in his/her software system, provides an accurate description of its complexity.

Another interesting observation is that the general consultants are aligned for the importance of the accuracy, explainability and visualisation, whereas the technical consultants are more aligned for the extensibility. This can be explained by the fact that the technical consultants are taking into account the development effort and the difficulty of extending the method to accommodate additional languages. On the other hand the general consultants are more interested in the presentation of the results because they

will use this information to communicate the results to the clients.

Another valuable piece of information which was extracted from these interviews is that the consultants are interested in the accuracy of the method, with different weight in False Positives than False Negatives. They can tolerate a reasonable percentage of False Negatives but it is important that the number of False Positives is minimal. They all agreed that the false positives should be less than 30% of the findings.

According to the average ranking of the RQs they are grouped, based on the MoSCoW method, as displayed in Table 4.1.

Table 4.1: Priority categorisation of requirements using MoSCoW, based on their importance according to the interviewees answers.

Requirement	Must	Should	Could	Won't
Accuracy	x			
Explainability	x			
Extensibility	x			
Initial range of languages		x		
Tool integration		x		
Visualisation		x		
Efficiency			x	
Filtering			x	
Exportability			x	
Universality				x
Language agnosticity				x

The RQs *Universality* and *Language agnosticity* are placed in the *Won't have* set because they were suggested only by one consultant. Thus they have the lowest importance value.

4.2 Rating criteria for requirements

Table 4.2 displays the requirements in order of importance (from high to low importance) based on the information obtained from the interviews. The horizontal groupings display each requirement, its metric in a five level Likert-type scale (Arnold et al., 1967) and for every level of that scale a statement of what the level represents. A description for each requirement is given in the paragraphs following the table. In this section the requirements which were listed under the *Won't have* category of the MoSCoW method, i.e., *Universality* and *Language agnosticity* are not included.

As can be seen from Table 4.2, one more category was added, i.e., *Prerequisites*. These

prerequisites have to be satisfied at least in the second level of our scale, in order to be able to assess the other requirements.

Table 4.2: Rating criteria for each requirement

Cat.	Requirement	Level	Level's description
Must have	Accuracy ($F_{measure}$)	*	0 ~ 20%
		**	21% ~ 40%
		***	41% ~ 60%
		****	61% ~ 80%
		*****	81% ~ 100%
	Explainability	*	only the developer of the method can explain the result
		**	reverse engineering on the analysis method is required to understand the results
		***	a technical consultant can interpret the result
		****	a business consultant with a weak technical background can interpret the result
		*****	a stakeholder or business consultant with no technical background can interpret the result
	Extensibility	*	not extensible, the methods is customised for a number of languages
		**	metamodels/parsers and detection patterns need to be defined for new languages
		***	only new metamodels/parsers need to be defined for new languages
		****	no metamodels/parsers used, but an extension mechanism is provided
		*****	no extension needed, the method is generic
Should have	Initial range of languages	*	1 ~ 3 languages
		**	4 ~6 languages
		***	7 ~ 9 languages
		****	10 ~ 12 languages
		*****	>12 languages
	Tool integration	*	the method is only theoretical (e.g., if it is only theoretical)
		**	the method is partly implemented as a stand alone application
		***	the method is fully implemented as a stand alone application
		****	the method is implemented in an integrated development environment
		*****	the method is integrated in a software analysis toolkit
	Visualisation	*	lacking visualisation
		**	console based text or graphical representation
		***	visual tool that shows dependencies as a grid
		****	visual tool that shows dependencies in one or more graphs
		*****	interactive visual tool for selecting dependencies from a graph for details

Continued on next page

Table 4.2 – Continued from previous page

Cat.	Criterion	Level	Level's description
Could have	Efficiency	*	needs more than 6 hours to finish the analysis for an ~1MSLOC system (Million Source Lines Of Code)
		**	between 3 to 6 hours to finish the analysis for an ~1MSLOC system
		***	between 1 to 2 hours to finish the analysis for an ~1MSLOC system
		****	between 30 minutes and 1 hour to finish the analysis for an ~1MSLOC system
		*****	less than 30 minutes to finish the analysis for an ~1MSLOC system
	Filtering	*	no filtering provided
		**	basic filtering capabilities according to the dependency type
		***	predefined category filters
		****	text filters
		*****	combination of text filters and category filters
	Exportability	*	not exportable results
		**	results exported in a method specific format
		***	results exported in one open or standardised format
		****	results exported in more than one open or standardised formats
		*****	results exported in more than one open or standardised formats in a configurable way
Pre-requirements	Maturity Testing	*	no evidence of testing
		**	1 ~ 2 case studies
		***	3 ~ 10 case studies
		****	> 10 case studies
		*****	tested in production environment
	Maturity Usability	*	un-deployable
		**	can be deployed and used by experts with deep knowledge of the method
		***	deployable and usable only by experts
		****	deployable only by experts but usable by software evaluators
		*****	deployable and usable by software evaluators
	Documentation	*	lacking documentation
		**	documentation that describes mainly deployment
		***	documentation that describes deployment and functionality
		****	documentation describing laconically each step of the method
		*****	full documentation with examples of usage

4.3 Must have requirements

Accuracy

From the compiled results from the interviews (Fig. 4.1), we can see that *Accuracy* is the main concern of the consultants. Therefore it has the first position in our ranking too. A method or a tool that extracts dependencies, belongs to the category of information retrieval methods. Especially for accuracy, there are published and commonly accepted metrics, which are explained in this paragraph.

Let C be the collection of all possible dependencies which is depicted with the outer ellipse in Fig. 4.2. Suppose that from all the items in C , only a set π with $\pi \subseteq C$ of them is “*relevant*” (or actual dependencies). Suppose also that a method that extracts dependencies detects and retrieves a set t for which $t \subseteq C$.

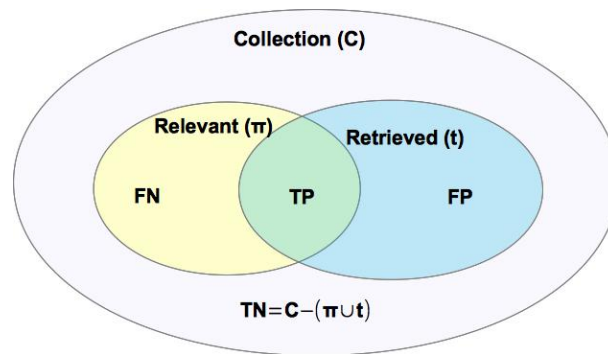


Figure 4.2: Metrics used to evaluate accuracy in the information retrieval domain. In our case *Collection* is the set of all possible dependencies that can exist in the system, *Relevant* is the set of actual dependencies in a project, and *Retrieved* is the set of dependencies that a dependency detection method retrieves. (FN=False Negatives, FP=False Positives, TP=True Positives, TN=True Negatives)

From Manning et al. (2009) we get table 4.3 which shows the possible combinations of findings.

Table 4.3: Possible dependencies

	Relevant	Non Relevant
Retrieved	True Positives (TP)	False Positives (FP)
Not Retrieved	False Negatives (FN)	True Negatives (TN)

From the same publication and from Alvarez (2002) we get the following classic metrics

for information retrieval.

“Recall (R) is the fraction of relevant documents that are retrieved”.

$$R = \frac{|TP|}{|\pi|} = \frac{|TP|}{|TP| + |FN|} \quad (4.1)$$

“Precision (P) is the fraction of retrieved documents that are relevant”

$$P = \frac{|TP|}{|t|} = \frac{|TP|}{|TP| + |FP|} \quad (4.2)$$

Although there was not a clear answer by the interviewees about the maximum number of FN, they all agreed that the precision should be higher than 70%.

Both measures are important for measuring accuracy. However, a single metric is more suitable for our comparison matrix because we can correspond each level of the ranking scale of the accuracy requirement to a range of values of that single metric. Van Rijsbergen (1979) introduces a metric that captures the trade-off between recall and precision. It is defined as the weighted harmonic mean of precision and recall and it is calculated as follows:

$$F_{measure} = \frac{(\beta^2 + 1) \times P \times R}{\beta^2 \times P + R}, \text{ with } \beta^2 \in [0, \infty] \quad (4.3)$$

β shows how many times recall is more important than precision. If $\beta = 1$ they are of equal importance. If $\beta > 1$, recall is more important and if $\beta < 1$, precision is more important. From the interviews with the consultants we learned that precision weights more than recall. Therefore $\beta = 0.5$ is selected which shows that precision is twice as important than recall. The metric for accuracy thus becomes:

$$F_{0.5} = \frac{(0.5^2 + 1) \times P \times R}{0.5^2 \times P + R} \quad (4.4)$$

Explainability

The second most important requirement, according to the interviews is explainability. It is of high concern for both technical and general consultants. By the term “*explainability*” we mean the property of the method that identifies how easily the method behaviour and results can be interpreted by the user. As explained in section 4.1, it is significant for two reasons:

- If the representation of the results of an analysis are self explanatory, it is cost-efficient for the software evaluator, who will not need time to spend in the interpretation of the results.
- If the meaning of the results and the way they were formed are clear to even non technical personnel, then the results are quickly accepted.

Extensibility

The third requirement, in terms of priority, is extensibility. The definition used for this requirement during the interviews was the ability of the method to be implemented or extended to include a currently unsupported language in a short time. The experts often encounter information systems which contain modules written in languages that have not been encountered before. Consequently, they have to update the method in a short time while keeping the trade-off with accuracy at a reasonable level. The metric we would like to use for this criterion is the number of days that is required to support a new language. However this question cannot be answered unless we try to extend each method ourselves, or find published information about it. Because no references were found to this kind of information, the type of approach is used as a criterion, namely how generic the method's algorithm is, e.g., does it use parsers or meta-models for each language, or a more generic approach?

4.4 Should have requirements

Initial range of languages

The range of languages that a method already supports is another important requirement. Especially for consulting companies that have to work daily with a wide variety of languages, a method that supports only a small subset of them is not adequate. It would require a lot of effort to extend such a method so as to support most of the languages of their clients' portfolios, even if the extensibility is high. The metric used here is the number of languages that the method supports.

Tool integration

This requirement shows how well the method under evaluation is implemented or integrated into a tool. It can vary from being purely theoretical with no implementation at all, or being implemented as a console/script based program, up to being fully implemented and integrated into a widely used development or analysis toolkit.

Visualisation

Visualisation describes the representation of the results of the analysis. Like the previous requirement, it can vary from a text based representation enumerating all the dependency findings, to a sophisticated interactive representation of the results as a graph. In the latter case, a software evaluator would be able to select a dependency which is represented by an edge and see more details. The reason that this is not so high in priority, is that the

focus of our research is not on the representation, but on quality of the findings and the extensibility of the method. The findings can also be merged with other evaluation tools, therefore the representation is important, but not crucial for the specific research.

4.5 Could have requirements

Efficiency

For efficiency we looked up the definition in the ISO25010 (2011) where *Performance efficiency* is described as the “*performance relative to the amount of resources used under stated conditions*”. It consists of three elements:

- *Time behaviour* - which concerns the response, processing times and throughput rates of a method.
- *Resource utilisation* - which concerns the amounts and types of resources used by the method.
- *Capacity* - which concerns the maximum limits of method’s parameters.

As the methods selected for review provide software tools for executing them, we plan to do their evaluation using a desktop computer, which has a moderate set of characteristics (4GB memory, 4x2.8GHz processing power) at the time this thesis was written. We plan to execute these methods on a large scale project, i.e., around one million source lines of code (MSLOC) and measure the time for the analysis to finish.

Filtering

Filtering is another feature that is useful for the methods. It refers to results filtering, i.e., how we isolate a group of dependencies of our interest, from the whole set of dependencies that the method retrieves. Filtering can be done by the different types of classification (see section 3.1.2). It can also be done using lexical keywords, or even regular expressions for the analysts with a stronger computer science (CS) background.

Exportability

A method for finding heterogeneous dependencies is intended to be used along with other methods in the context of comprehending software. These methods produce results in a human readable format that aims for better understanding of the software program at hand (e.g., in control flow analysis, control flow graphs are produced). It is valuable when the results of each method are combined in a unified representation because it saves time and helps to have an overall image of the program’s complexity. This is why exportability of the results is one of the requirements for the methods of

our review. Exportability describes how easily the results of a method can be used from other methods or frameworks, thus improving the interoperability with them. Examples of open formats for direct graphs representations are GraphML, GXL, XMI etc. The metric is the number and the type of formats in which the results are exported. It is placed last in our ranking from the interviews because, as mentioned by the experts, the tool integration may overlap this feature.

4.6 Prerequisites

This section describes three prerequisites that an existing method has to satisfy, up to a degree, in order to be capable of evaluating the method against the Must, Should, and Could have requirements.

Maturity is an important prerequisite that contributes to the quality of a method. Usually the lifecycle of a method involves the conception of the initial idea, or the algorithm, then its implementation, followed by iterations of testing, evaluation, and refinement. When the algorithm and its implementation are ready, more effort in improving usability and other lower priority attributes is invested. With this requirement the maturity state of the methods is evaluated. That is why this prerequisite is split in two: application and usability.

Maturity - Application

This prerequisite shows the extent to which the method has been applied. The lowest level is that the method has not been tested, or that no evidence of usage is found. Other levels include the execution of the method on a number of case studies, and the highest level is that the method is applied in a production environment.

Maturity - Usability

Adapting the definition of ISO25010 (2011) to our case, Usability is the degree to which a method can be used by the evaluators to achieve specified goals with effectiveness, efficiency and satisfaction. For our evaluation, we focus on the ease of deployment and usage of the methods and the provided tools.

Documentation

The next prerequisite added is “Documentation”. It describes the quality of the documentation of the method, how its steps are described, if there are examples of usage so

as to make a software evaluator able to use it. If a method has very complicated steps that are not explained, or if the supporting tool of that method is very difficult to understand due to lacking documentation, then the method's usability decreases. However, a well documented method leads to improved usability, maturity, and extensibility of the method.

4.7 Comparison matrix

The requirements and their metrics which were defined in the previous section, provide the means for evaluating the state of art methods for detecting cross-language dependencies. A comparison matrix is an informative and descriptive way of visualizing the differences among the reviewed methods. The names of the state of the art methods are the column headers of the table and the row headers are the requirements. The contents of the cells contain the score of each method for the corresponding requirement. A question mark in a cell shows that the score could not be calculated or deducted, or that there is uncertainty about it. The comparison matrix is displayed in Table 4.4.

Source Navigator NG (SN) has a high score in many requirements because it supports many languages and it provides a documented way of extending it for more languages. We were able to install it and conduct an analysis for three supported languages (Java, C, PHP). Files from each language that we created were used as input. C and Java files declared fields and methods, and the PHP files called the same method names and used fields with the same names. The analysis showed dependencies within each language but not any cross-language dependency. The analysis was run on other three multilingual programs, but without success. Since we do not know what are the dependencies the SN looks for, we are not confident about its score. Moise and Wong (2005) did not find any examples for such use either. In the same paper they extended SN to find Java and C++ dependencies through Java Native Interface. One year later they extended it further for finding dependencies across C++ and scripting languages as Perl and Python. Source Navigator NG is still active but it does not have a big developer community¹. For the above mentioned reasons, SN was excluded from the comparison matrix.

We tried to execute all the methods which are shown in the matrix. Some of them were deployable and usable while others do not provide the proper documentation and could not be used at all (e.g., PAMOMO, GenDeMoG). Even for the methods that could be used, we were not able to find the right information, in a reasonable time-frame, to measure every requirement of our table (Source Navigator NG, Eclipse MoDisco, PAMOMO, GenDeMoG). For the evaluation of those cases the information obtained from the related publications was used. The low maturity ranking of most of the methods depicts this problem. Moreover, for the methods that were applicable, the language set was different. Therefore any comparison would not be reliable. For these reasons

¹http://developer.berlios.de/project/stats/?group_id=8334

Table 4.4: Comparison matrix of the state of the art methods according to their rating on the requirements set we defined.

Priority	Requirement	MoDisco	PAMOMO	GenDeMoG	Moose	DSketch
Must have	Accuracy	?	?	?	?	?
	Explainability	?	**	***	***	***
	Extensibility	**	**	**	***	*
Could have	Initial range of languages	*	*	***	***	*
	Tool integration	****	****	****	*****	****
	Visualisation	***	***	****	****	***
Should have	Efficiency	?	?	?	?	?
	Filtering	****	?	?	***	****
	Exportability	***	?	***	**	*
Pre-reqs	Maturity - usability	***	**	*	****	***
	Maturity - testing	*	**	**	***	**
	Documentation	*****	**	*	*****	****

question marks were put in the rating of requirements *Accuracy* and *Efficiency*.

From all the methods, only GenDeMoG provided a sophisticated accuracy metric. The authors of the method analysed a heterogeneous application (the Apache Ofbiz enterprise automation suite²); and by manually inspecting all the dependency patterns that they used in the analysis, they determined that no False Positives were retrieved. However they did not check for False Negatives. The GenDeMoG is a very thorough approach but it lacks documentation so it was difficult to test its applicability.

The method which ranks best according to our requirements is the one offered by the Moose project. It is the most mature from the reviewed methods, it supports a relatively high range of languages and it provides a framework for extending it further (in terms of new languages, results exporting, visualisation, etc.). We were able to install it and conduct an analysis with it without problems. It also provides detailed documentation describing its usage and its extension possibilities.

We also observe that all the methods score low in explainability, which is a key requirement of the evaluators. Another interesting fact is that the combination of the criteria *Initial range of languages*, and *Extensibility* makes all the methods rank low. We take into account this average because our target is to have a method that supports a considerable number of languages after an investment of a reasonable amount of time and cost.

²<http://ofbiz.apache.org/>

Eclipse MoDisco has invested a lot of effort building an infrastructure which is able to support (potentially) many languages and there is a large active community working on it. It uses meta-modelling techniques to map each different language structure to a common meta-model for all languages. Patterns have to be provided by the user to find the cross references among the attributes of the different models. This results in its low maturity ranking in our comparison matrix. However it has a serious potential for becoming a strong framework for source code analysis.

All the methods reviewed, tend to use a lot of language specific information in order to extract dependencies. Except DSketch all the methods which use meta-modelling techniques require a parser or a modeller to be constructed for supporting new languages and both of these components require detailed knowledge of the language's syntax and grammar. This leads them to score low in extensibility. For the company where this thesis is conducted, this is a severe drawback because the systems the company has dealt with in the past, contain in total more than 70 different languages. The reviewed methods offer support for less than nine languages initially and in combination with the low extensibility, it results in great cost and effort if we decide to use them for a broad range of systems. Therefore, the approach of meta-models has to be rejected until a fast and automated way of creating meta-models from programming languages is defined. The DSketch method may not use a parser or a modeller, but it is still based on a specific language grammar, and it does not provide an elaborate extension mechanism.

For these reasons we decided not to base our research on one of the current approaches, but to introduce a new method. As stated in our objective, we focus on the cost-effectiveness while keeping the accuracy in acceptable levels. It is a challenging trade-off which is addressed in the next chapters.

Chapter 4. State of the art evaluation

Chapter 5

Proposed method

This chapter describes the longest phase of the thesis because it involves the gradual development and evaluation of the proposed method. The following sections provide information for each step of method creation.

5.1 General description

Our goal is to extract dependencies in heterogeneous software systems using the definition in 3.1.1. Patterns and algorithms are developed to extract the dependency types which were mentioned in chapter 3. A generic solution is difficult to find, therefore language specific information may be needed. Our intend is to minimise the language specific parts of our algorithm.

A method prototype has to be built to test the developed method. For this, the classic steps of every software project take place: design of the prototype, implementation and testing. After the testing of each algorithm, we experiment with different parameters to evaluate the algorithm. The definition of the algorithm and the prototype development is an iterative process because the algorithm needs to be implemented, tested, evaluated, and refined.

In the coming sections our approaches and the algorithms involved are described. For each algorithm an explanation for its selection and the observations from its evaluation are given.

5.2 Preparation for the evaluation process

5.2.1 Case study

Taking into account all the decisions in section 2.5.2, a sample application from the Spring webflow framework or SWF¹ was selected as our case study project. The application is called “*booking-faces*” and it is a web based hotel booking application. The source code can be downloaded from the Spring community website² by navigating to Spring webflow and then to version “2.1.0.RELEASE”. The Spring framework is a framework for building layered Java enterprise solutions. SWF is an extension to this framework which enables the use of flows and transitions from one web page to another.

The project contains only 98 files of which the source files are only a few. An analysis for the physical lines of code using Al Danial’s³ *cloc* tool produced the following output:

```

http://cloc.sourceforge.net v 1.53 T=0.5 s (92.0 files/s, 6412.0 lines/s)
-----
Language          files      blank      comment      code
-----
CSS                14         213         199          961
HTML              10          24          21          558
Java               8           8          57          476
XML               12          76          45          357
SQL                2           3           0           62
-----
SUM:              46         470         322         2414
-----

```

It has Java, XML, HTML, SQL and CSS as main languages. The technologies it involves are:

- Java
- Java Persistence (Component for database handling)
- Spring Webflow (Component of Spring Framework, responsible for the flow and the transitions between web pages)
- Java Server Faces (Presentation layer component for rich web pages)

The consultants did not show interest in dependencies in which CSS files are involved, thus they are excluded from the files set among which the dependencies are detected.

¹<http://www.springsource.org/spring-web-flow>

²<http://www.springsource.org/download/community>

³<http://cloc.sourceforge.net>

5.2.2 Experiment setup

The ground truth method and the case study project have already been selected. The next step is to define the nature of the dependencies we are looking for, namely the *direct*, *framework*, or *indirect*. (The reader can refer to section 3.1.2 where these terms are explained.) It is useful to check the types of the dependencies as well, because looking explicitly for certain dependency types (e.g., database access dependencies), increases the accuracy of the manual checking.

The second step is to define the languages or the technologies across which, files are dependent. In our case Java, SQL, XML and HTML/XHTML were selected.

Now, there is enough information to define the collection of file combinations within which the dependencies are extracted.

In the *booking-faces* project, CSS(.css), Java Manifest(.MF), photo(.jpg, .png, .gif) or Maven (.pom) files are excluded, so of the 98 files in the project, only 32 are involved in our dependency checking. Files under the test directories of the project are also omitted. If two files are dependent in both directions, this dependency is counted once. The dependencies which are among files of the same language are also filtered out.

The last step before we start evaluating our proposed method, is to execute the ground truth method on the case study project. Table 5.1 displays the information retrieved with a manual check for dependencies.

Table 5.1: Ground truth data

Attributes	Value
Direct dependencies	5
Framework dependencies	39
Total relevant dependencies (π)	44
Collection (C)	367
Non relevant dependencies (<i>Total negatives</i>)	323

The reader can refer to section 3.1.2 as well as to section 4.3 where the terms in the table are explained.

5.3 SIG framework

In section 1.5 it is mentioned that SIG assesses the quality of the software which is used or provided by their client companies. The big number of different programming

languages they encounter (over 70) has created the need to build a method and a tool (Software Analysis Toolkit or SAT) to analyse these systems. SAT provides an analysis base framework on top of which analysis tools for different languages can efficiently be built. SAT is able to produce accurate analyses and dependency results for most of the 70 programming languages which they have encountered. For example, SAT can generate, for the Java language, the dependency graph of a system, with method level granularity, and with metrics for every element of the maintainability quality standard (ISO25010, 2011). It does that by automatically calculating metrics (e.g., lines of code and McCabe’s complexity (McCabe, 1976)).

5.4 Infrastructure

This section describes the infrastructure of our prototype. It is illustrated in Fig. 5.1. The prototype needs an infrastructure which has a lot in common with SIG’s framework. Given a root directory which contains the modules of the system to be analysed, SAT creates a graph of the files and directories and populates it with more information, for instance the programming language that each file belongs to.

The main idea is to have a set of layers on top of SAT, where the source code can be searched for multiple dependency types, using the “enriched” file graph which is generated by SAT.

We consider that a dependency type describes a directed relationship and even if it is not directed it is treated as directed with a random direction. In section ?? it is mentioned that if two files are connected with two dependencies of different direction, this dependency is counted once. This gives the flexibility to detect directed and non-directed dependencies in a uniform way. Therefore we search first in the source files to extract the definition of a fact (*callee fact*), i.e., a code snippet, and then search again in the files for other facts which depend upon the callee facts. The dependent facts are called *caller facts*. Our intent is to build an inventory of dependency types where each type adheres to the design just described. Each type has the caller and the callee patterns which are used to detect the corresponding facts in the source code for the specific dependency that the type describes. The infrastructure provides the ability for each pattern to be implemented generically and if necessary, language specifically. For the language specific way, a language handling component is responsible for implementing specific patterns for each language.

During the analysis, a dependency graph is created. Then edges are created from the extracted facts to the callee files, and then from the caller files to the facts. Using that information edges are created from the caller files to the corresponding callee files if there is a node connecting both of them.

After the analysis finishes, there is a filtering component that is responsible for filtering

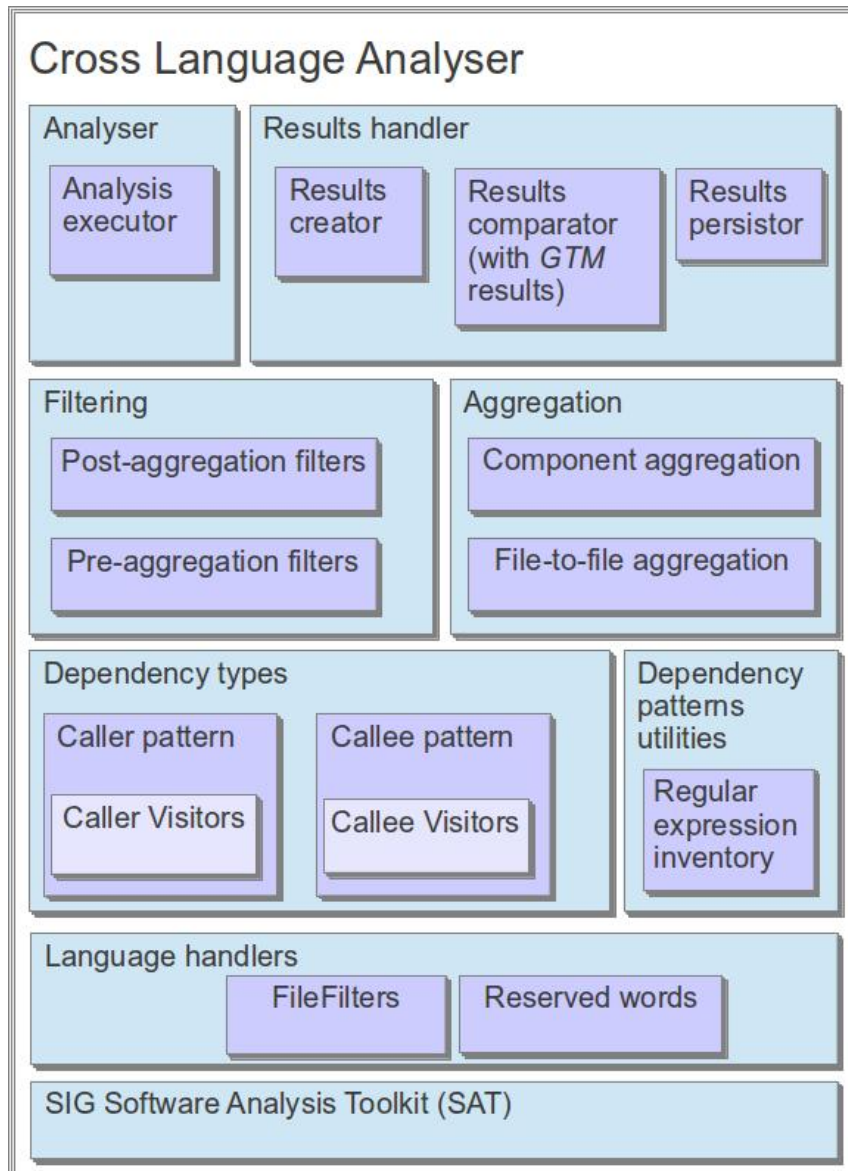


Figure 5.1: Architecture diagram of the prototype of the proposed method. The boxes represent the components (i.e., packages) of the prototype. A component can contain sub-components which are represented with inner boxes of different colour.

out dependencies that are not relevant, e.g., edges where the caller and callee files are the same, or they belong to the same language.

Moreover, the aggregation component of our infrastructure which uses SIG's framework as well, is responsible for aggregating the dependencies to a component level. A component can be a directory or another type of grouping. If a system has more than a few files, presenting the results without aggregating the dependencies is not very useful.

An important part of our infrastructure is the presentation and the persistence of results. The results are automatically compared with the ones from the ground truth method, and then all the information is stored into a database for further analyses. The results, apart from the metrics discussed in the evaluation process, (recall, precision and $F_{0.5}$) contain detailed information about the dependencies, i.e., the caller file, the callee file, the name of the dependency type and source facts that were detected by the two patterns. The analysis can be done for multiple dependency types simultaneously.

This infrastructure allows for the extension of a dependency type to support more languages. Apart from being used for our experiments, the resulting dependency graph gives an informative overview of the complexity of the system.

5.5 Pattern based approach

Description

The infrastructure is designed in a way that the analysis can be done for a number of dependency types. The interviews with the experts showed that the dependency types they are most interested in, are systems that involve modules which communicate through databases, files and web services.

5.5.1 Database dependency type

DB entity reference in embedded SQL

We decided to start developing our method by detecting dependencies among database and host languages. Van Den Brink et al. (2007a) explored the dependencies among databases and host languages in which SQL queries are embedded. They focus on Visual Basic, PL/SQL and COBOL and they manage to reveal the complexity of the involved languages with high accuracy. Although this work is remarkable, it has the disadvantage that it cannot be efficiently maintained for and extended to many languages, because of its language specific information. That is why a less accurate but more flexible approach is followed.

The first relationship extracted is the interactions among tables of a database schema and a host language. Therefore our callee pattern should reveal the table name and our caller pattern should reveal the references to it. In order to analyse the database and extract the table names, the database schema is exported to an .sql file which contains all the SQL entities creation declaration i.e., schema, tables, functions, procedures, triggers, etc.

Callee pattern

For the callee pattern, we searched the syntax definition of SQL for table creation declarations among the commonly used relational database management systems (RDBMS), e.g., MySQL DB, PostgreSQL and Oracle DB. The conclusion was that there are two main ways to find table creation statements in SQL code: by parsing the .sql file using the full SQL syntax and by lexical text matching using regular expressions. Because the table names only need to be extracted from the .sql file, no parsing of the SQL creation statement is required. Our hypothesis was that with a regular expression the table names can be extracted, covering the majority of cases. In the end, we came up with the regular expression in (5.1).

$$(?<=create\ table)(\s+\w+\.? \w+\b) \quad (5.1)$$

The callee pattern worked with high accuracy on the different files considered, from all three RDBMS systems. With the above regular expression, the schema name with a full-stop is also obtained if the table name has the form "SCHEMA.TABLENAME". It is very easy to extract only the table name. For each .sql file a node is created for every table name it contains and an edge is created from the table name to the file. The result is that for every .sql file a list of table names is associated with it.

Caller pattern

For the caller pattern, the algorithm was constructed by searching in every non-sql files for words that match any of the table names which already exist in the graph. If a match for a table is found, then an edge in the graph from the non-sql file to the table node is created. Fig. 5.2 shows a sample graph at that stage. In this example, three Java files and two SQL files are included under the directory src/demoApplication. The two SQL files create the same tables (USER and PERMISSIONS) but they have different notation, because they are targeted to a different RDBMS. The UserManagementDao.java file has embedded SQL referring to those tables and the other two Java files have irrelevant content. Fig. 5.2 shows how the nodes *USER* and *PERMISSIONS* are connected with the corresponding source files.

An in depth investigation showed that the table names were not only matched by the embedded SQL statements, but also by object names. To correct this, the caller pattern

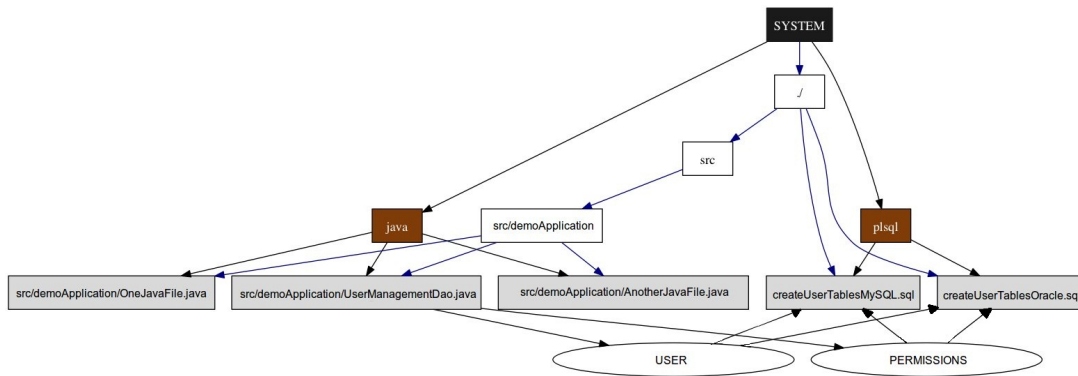


Figure 5.2: Sample dependency graph

was adapted to search for table names, not anywhere in the file, but only in quoted text, with the rationale that embedded SQL can usually be found in string assignments. Because quoted text can be declared in different ways across programming languages, only text within double quotes is searched. We extracted the quoted strings with the regular expression shown in (5.2) and searched in these strings for the table names. For other languages where the string assignment declaration is different, a different (but similar) implementation of the pattern can be created.

$$(?:\" ([^\"]+)\" | (\\S+)) \quad (5.2)$$

To improve the caller pattern from the extracted quoted strings, the ones that did not contain any SQL reserved word or phrase in them are excluded, such as *INSERT INTO*, *SELECT FROM*, *UPDATE* and *DELETE FROM*. In this manner, the probability of matching a SQL statement increases, but it is possible, that these phrases and words are part of a physical language string.

The next step was to extend the facts that are captured with the callee pattern. Besides extracting only table names, SQL functions, procedures, triggers and other SQL entity types are extracted. “create or replace” was added in our case-insensitive search string to cover more cases.

Evaluation

Although the extraction of entities from .sql files had 100% accuracy, the matching of the entities in the embedded SQL statements produced False Positives. Quoted strings may include SQL reserved words, but that does not make them SQL statements. The more we were trying to reduce the False Positives, and search with more structure the quoted texts, the more language specific information (SQL grammar & syntax) we introduced to the algorithm. Consequently, this lowers the extensibility and the explainability of

the method. The approach of extracting embedded SQL code in host languages was researched by Van Den Brink et al. (2007b). This paper was the outcome of the main author’s master thesis which was conducted also in SIG. The co-authors who are also working at the same company, informed us that this approach was highly accurate, but was not easily extensible for other host languages, thus it was not adopted by the company.

The experience obtained with the database dependency type showed us that it is very difficult to develop patterns for every dependency type and language without involving a great amount of language specific information.

5.5.2 File I/O dependency type

The different ways in which languages interact through text files of a file system were also examined, the so called file input output (I/O) dependency type. A list of code facts which are responsible for reading, writing and appending files was created for the languages Java, C, C++, C#, VBNET, PHP and TCL/TK. We were surprised by the variety of ways that exist for opening and reading or writing to a file. Modern languages use *stream readers* while others a simple open command. Some code snippets are shown in Table 5.2.

Table 5.2: Examples of file handling

Language	File handling example
Java	<code>new BufferedReader(new FileReader("file.txt"))</code> <code>new FileInputStream("file.txt")</code>
TCL/TK	<code>set out [open "file.txt" r]</code>
C	<code>FILE *fp; fp = fopen("file", "r");</code>
C++	<code>ifstream infile("file.txt")</code>
C#	<code>new StreamReader("file.txt")</code>
PHP	<code>\$file=fopen("file.txt", "r");</code>
VBNET	<code>Dim Filename As String = "file.txt"</code>

The examples in the table are just a few ways in our list of accessing files in those languages. We wanted to point out this diversity in order to justify our decision not to search for patterns, but to seek for an alternative solution for retrieving these dependencies.

5.6 Token based approach - Ad-hoc filtering

As an alternative to the pattern based, a more generic, token based approach is proposed. Here the basic algorithm is proposed, on which we elaborate to improve its accuracy.

5.6.1 Basic algorithm

With the following algorithm files from different programming languages are associated, according to similar tokens which they may have. The basic algorithm is shown in Algorithm 1.

Algorithm 1 Basic algorithm of the token based approach

```

1: create a graph of the system's source files
2: for each file in the graph do
3:   replace special characters in the file's content with spaces
4:   tokenise the text by splitting it on white space
5:   extract all the words-tokens
6:   for each word-token do
7:     lookup in the graph if a node with the name of the word exists
8:     if word-node does not exist then
9:       create a node with the same name as the word
10:    end if
11:    create an edge from the file to the word-node
12:  end for
13: end for
14: for each word-node do
15:   if node is connected with fewer than two files from different languages then
16:     remove node
17:   end if
18: end for

```

The following details of the algorithm were omitted because we want to give just the basic idea while keeping it readable in this document:

- A special character is any character except a letter, number, and/or underscore.
- During the tokenising, strings that consist only of numbers are omitted.
- Tokens with less than one character are also omitted.

The next paragraphs describe the steps performed to improve the accuracy of our algorithm. All the experiments in this section were conducted using the "booking-faces" case study project, and the results of our algorithm are compared with the ones that were produced using the ground truth method on the same project. The coming paragraphs have the following structure:

- Description of the method to increase accuracy, based on the aforementioned algorithm.
- Evaluation of the results.

In this document only the main results from our experiments are reported. Each experiment has an identification number (Id) and for every new experiment the Id is incremented by one. For every experiment, the result is added to the table as a new row where the first column is the Id of the experiment.

5.6.2 Base case

Description

The basic algorithm performs a “brute force” words matching. No filtering is applied on the graph as this is the base case of our experiment.

Evaluation

Table 5.3 shows the results from the first experiment using the token matching algorithm.

Table 5.3: Results from token based approach - Ad-hoc filtering: Base case

Id	Recall	Precision	F_{0.5}	Description
1	100%	11.6%	14.1%	basic algorithm

The results show that although the recall is perfect, the percentages of precision and F_{0.5} are very low. This is not surprising because the simplest form of the algorithm was used. In the next steps ways to filter out the False Positives are considered.

5.6.3 Filtering out language reserved words

Description

The first thing to notice by examining the False Positives is that many of them are created due to words which belong to the language grammar, or its libraries and are also common in more than one programming language, e.g., *for*, *return*, *if*, *else*, *true*, *false*, *int*, *long*, *import* are examples of words that are used in more than one language. To filter out these dependencies a list of the reserved words for every supported language was created. The words in the list are excluded during the extraction of the words from the source files. For example, for XML, the following list was created: *xml*, *version*, *xmlns*, *xsi*, *schemaLocation*, *noNamespaceSchemaLocation*, *type*, *LastUpdate*, *encoding*, *utf*, *utf8*, *iso*, *element*, *sequence*. The W3C’s XML Schema definition⁴ was used to acquire commonly used words for XML files.

We also noticed that specific .xml files and .properties are connected with all the .java files, e.g., log4j.xml and log4j.properties. These are configuration files that are respon-

⁴<http://www.w3.org/XML/Schema>

sible for logging information during run time to text files⁵. Because they do not belong to the dependency types we are looking for, these files were filtered out.

Evaluation

Table 5.4 shows the results from this experiment.

Table 5.4: Results from token based approach - Ad-hoc filtering: Filtering out reserved words

Id	Recall	Precision	F_{0.5}	Description
1	100%	11.6%	14.1%	basic algorithm
5	100%	13.3%	16.1%	filtering out reserved words (Java, SQL, XML)

Although the recall remained to 100% and the F_{0.5} increased, we still observed that there were still FP created by keywords or common words from the libraries of the languages, e.g., *integer*, *boolean* and *Date*.

In order to tackle this problem the Java excluded word list was extended with all the package and the class names from the Java Development Kit (JDK)⁶. A program in Java to get the list of these words was created. Despite increasing precision, the performance of the algorithm dropped by at least 50%. We also observed that there are many JDK class names that are commonly used as identifiers (without using the specific JDK class), and they should not be excluded during the dependency detection. Therefore this solution was rejected.

Instead of including all the words of the JDK, only the ones that are under the package *java.lang* were used because the false positives were mostly from these classes. The speed of the algorithm increased significantly, but at the expense of precision. Table 5.5 shows the results.

5.6.4 Filtering out project specific words

Description

The precision is still low from the previous experiment (less than 70% that the consultants require). Another group of FP which contributes to these levels of our metrics is created because of project specific words which appear in a large percentage of files, e.g.,

⁵<http://logging.apache.org/log4j>

⁶<http://www.oracle.com/technetwork/java/javase/overview/index.html>

Table 5.5: Results from token based approach - Ad-hoc filtering: Filtering out language reserved words. The classnames under the java.lang package are added to the java excluded words list.

Id	Recall	Precision	F_{0.5}	Description
1	100%	11.6%	14.1%	basic algorithm
5	100%	13.3%	16.1%	filtering out reserved words (Java, SQL, XML)
6	100%	14.4%	17.4%	filtering out JDK words
7	100%	13.5%	16.3%	filtering out java.lang words

the basic package declaration in Java files which happened to have common words with the namespaces for many XML files as well. A filter to cut off these words was added to our algorithm.

Evaluation

Table 5.6: Results from token based approach - Ad-hoc filtering: Filtering out project specific words

Id	Recall	Precision	F_{0.5}	Description
1	100%	11.6%	14.1%	basic algorithm
5	100%	13.3%	16.1%	filtering out reserved words (Java, SQL, XML)
6	100%	14.4%	17.4%	filtering out JDK words
7	100%	13.5%	16.3%	filtering out java.lang words
9	100%	15.7%	18.9%	filtering out project specific words

Table 5.6 shows that recall is still at 100%, and F_{0.5} is increased by 2.6%. The problem is that this solution must be customised for every new project.

5.6.5 Tailored tokenising of XML type documents

Description

A precision of 15% is still low. By observing the FP dependencies again, we found out that some of them were created by the elements and parameters from the XML tags as in the lines of the following sample files:

booking-flow.xml (Spring webflow XML language):

```
<view-state id="enterBookingDetails" model="booking">
```

enterBookingDetails.xhtml (a JSF page, variant of JSP):

```
<h:inputText id="checkoutDate" value="#{booking.checkoutDate}">
```

web.xml (a file that configures a Java based web application):

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/config/web-application-config.xml
  </param-value>
</context-param>
```

The above examples show that the facts “id” and “value” create false dependencies among these files because they refer to irrelevant concepts.

To solve this, the base algorithm was implemented especially for XML type languages, to extract words only from the value elements (e.g., XML, XSD, HTML, XHTML and Spring framework related files). There are two types of value elements: the ones that are within quotes within a tag (e.g., “*enterBookingDetails*” in the first example) and the ones that are enclosed by opening and closing tags. (e.g., “*/WEB-INF/config/web-application-config.xml*” in the third example).

We manage to capture for every XML type file the content between quotes with the regular expression 5.2 that was presented earlier. Then, we capture for the same file all the content between tags (including the tags) with the regular expression 5.3 and delete it.

$$\langle (.\|\\n)+? \rangle \quad (5.3)$$

The algorithm is shown in Algorithm 2.

Algorithm 2 Capturing values from XML files

```
1: for every file do
2:   capture the quoted text
3:   for every quoted string do
4:     tokenise
5:     create or append the list of words for the file with the tokens
6:   end for
7:   replace the text between tags (including tags) with spaces
8:   tokenise the rest of the content
9:   append the list of words with the tokens
10: end for
```

Evaluation

Table 5.7: Results from token based approach - Ad-hoc filtering: Tailored tokenising of XML type documents

Id	Recall	Precision	F_{0.5}	Description
1	100%	11.6%	14.1%	basic algorithm
5	100%	13.3%	16.1%	filtering out reserved words (Java, SQL, XML)
6	100%	14.4%	17.4%	filtering out JDK words
7	100%	13.5%	16.3%	filtering out java.lang words
9	100%	15.7%	18.9%	filtering out project specific words
11	97.8%	21.5%	25.5%	extracting value elements for XML
13	97.8%	25.3%	29.7%	filtering out JSP EL, DTD & DOCTYPE words

With a slight decrease in recall a noticeable improve in precision was achieved. We are aware of the fact that the language syntax was used, but XML is a general category of languages rather than a language itself, therefore we believe we are not customizing the method to our case study.

In addition, by extending the excluded word list with the reserved words for the expression language (EL) for JSP and JSF and also adding the words DOCTYPE and DTD, the F_{0.5} increases to 29.7%(see entry 13 in Table 5.7). These two words, are in the header of many JSF and XML pages, therefore contributing to the high FP rate.

5.6.6 Evaluation of token based approach with Ad-hoc filtering

Despite keeping recall in exceptional levels, precision is still not satisfactory. We remind the reader that the interview sessions with the consultants showed that high precision is more important than recall. On top of that, for every refinement of the algorithm, language specific information had to be added. Therefore the explainability and the extensibility of the method decreased with every refinement.

5.7 Token based approach - Statistical filtering

In this section another branch of our token based approach is discussed. The reserved words filtering is removed and instead, a series of statistics based filters are applied. Our goal is to make the algorithm more generic, thus more extendible and comprehensible.

The following filters apply simple statistical analysis to files and their tokens. For the variables that are used in each analysis, the corresponding thresholds are given as input to those filters to cut off tokens or dependencies. For every filter, experiments are conducted with many different threshold values, in order to get the best results. In the next paragraphs the rationale for the creation of each filter is described and the accuracy of the method after experimenting with each filter separately (i.e., removing the previous filters) is evaluated. Since each filter is parametrised, the values of the accuracy metrics are presented with a chart, as it shows in a more comprehensible way the improvement or deterioration of the metrics.

5.7.1 Filtering out frequent words

Description

We observe that language reserved words appear with high frequency among the total words of all the files in the project. This led to the creation of the first filter of this category which calculates all the frequencies of unique tokens within the project. This is done by creating a map of all the unique tokens that are extracted from all files, and by counting the number of times each token is found. Our goal is to remove words that appear frequently, like *false*, *true*, *integer*, etc. All the reserved word filters that were presented in the ad-hoc filtering approach are removed, and instead the tokens frequency filter are applied. The input to the filter is a frequency threshold. This means that words that appear in the whole project with higher frequency than the threshold value, are removed, and are not involved in the detected dependencies.

Evaluation

In the *Xaxis* in Fig. 5.3, the frequency thresholds do not have the same intervals. The values in the figure were selected after experimentation with a wide range of thresholds, because they better represent the changes in the metrics visually (e.g., the metrics have the same values for the thresholds from 100% to 6%).

Table 5.8: Token based approach - Statistical filtering: Filtering out frequent words

Id	Recall	Precision	F_{0.5}	Description
16	97.8%	16.7%	20%	Filtering frequent words, Threshold=6%

The chart also shows that the recall decreases for threshold=1% while the precision remains stable. In order to explain that, the FP and the FN were examined. We

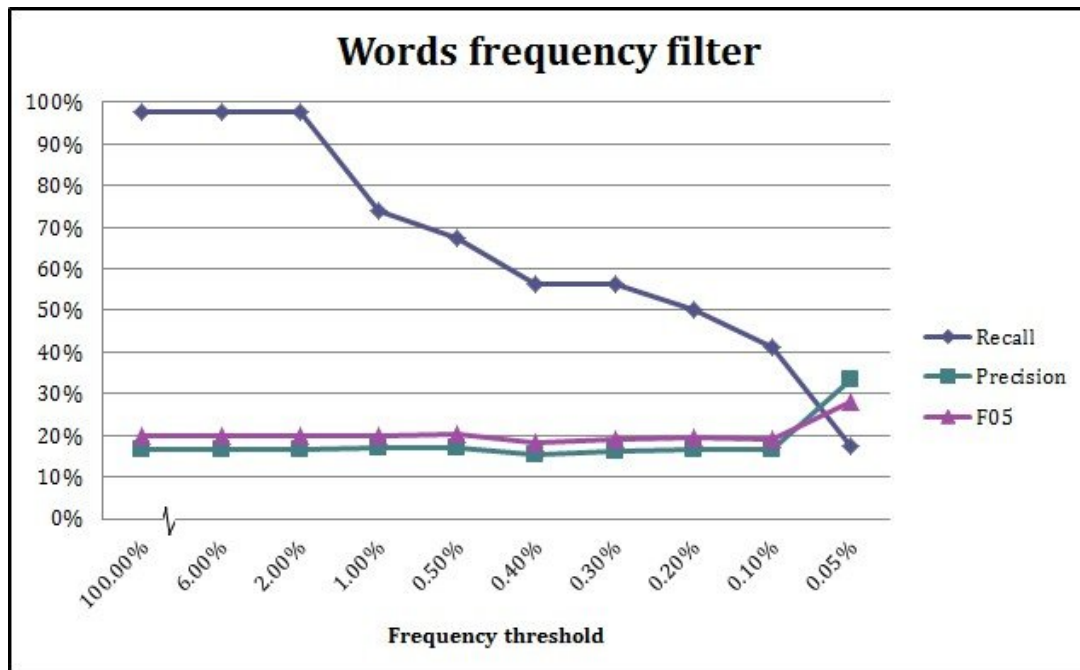


Figure 5.3: Token based approach - Statistical filtering: Filtering out frequent words

observed that common words within a language (which create FP) were not removed (e.g., private, string, html). This is due to some words not being keywords or common words for every language.

On the contrary words (e.g., like *hotel*, *booking*, *price*, etc.) which were part of the actual dependencies were filtered out because they were found in many places in the source files.

5.7.2 Filtering out frequent words per language

Description

In order to improve our results, the previous filter was modified to calculate the frequencies of the words per language. In this way, we hypothesise that a word that is common for one language but not for others will still be considered frequent according to the input threshold.

Evaluation

The results of the filter are shown in Fig. 5.4. With the frequent words per language filter, the results are nearly the same when the recall is around 100% but they are slightly better when the recall drops to around 50%.

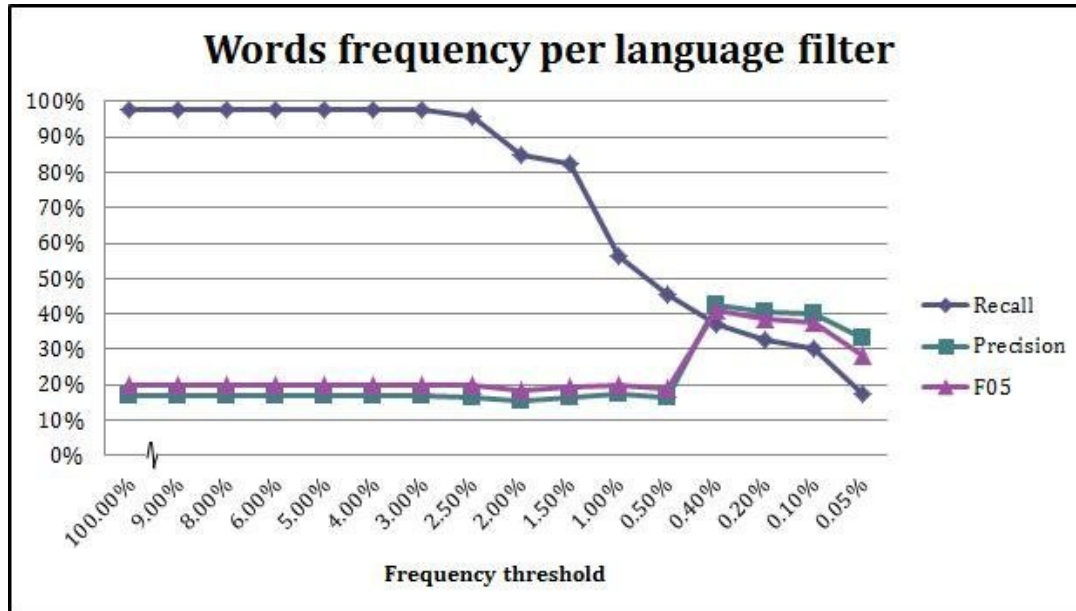


Figure 5.4: Token based approach - Statistical filtering: Filtering out frequent words per language

The same number of FP can be explained by the fact that sometimes the source files contain a big number of statements which do not necessarily include language specific words. An example that reveals this problem is a system that has many Java files which have a high number of lines of code (LOCs). As such, for this system, for the Java language, the words *class*, *interface*, *extends*, *implements*, *etc.* will not have high values of frequency, hence they will not be removed.

We also observe that for threshold=0.4% (see Table 5.9) there is a decrease in recall but the $F_{0.5}$ increases greatly (to 41.3%).

5.7.3 Filtering out frequent words per file and language

Description

In order to overcome the problem that was described in the previous paragraph, the *Words per files filter* is introduced. This filter calculates the frequency of the words which

Table 5.9: Token based approach - Statistical filtering: Filtering out frequent words per language

Id	Recall	Precision	F_{0.5}	Description
16	97.8%	16.7%	20%	Filtering frequent words, Threshold=6%
30	97.8%	16.7%	20%	Filtering frequent words per language, Threshold=3%
36	37.0%	42.5%	41.3%	Filtering frequent words per language, Threshold=0.4%

appear in different files of the same language. It collects all the unique tokens(words) for the specific language and then for each word it counts the files containing it. The filter calculates for each language and for each word the fraction:

$$\text{Word freq. per files} = \frac{\text{Number of files from a language that contain the word}}{\text{Total number of files from this language}} \quad (5.4)$$

Evaluation

Fig. 5.5 and Table 5.10 show that for threshold=60% a small improvement in the precision and the F_{0.5} is achieved while keeping the recall at very high levels (97.8%).

Table 5.10: Token based approach - Statistical filtering: Filtering out frequent words per file and per language

Id	Recall	Precision	F_{0.5}	Description
16	97.8%	16.7%	32.6%	Filtering frequent words, Threshold=6%
30	97.8%	16.7%	32.6%	Filtering frequent words per language, Threshold=3%
36	37.0%	42.5%	41.3%	Filtering frequent words per language, Threshold=0.4%
55	97.8%	18.4%	22%	Filtering frequent words per files, Threshold=60%

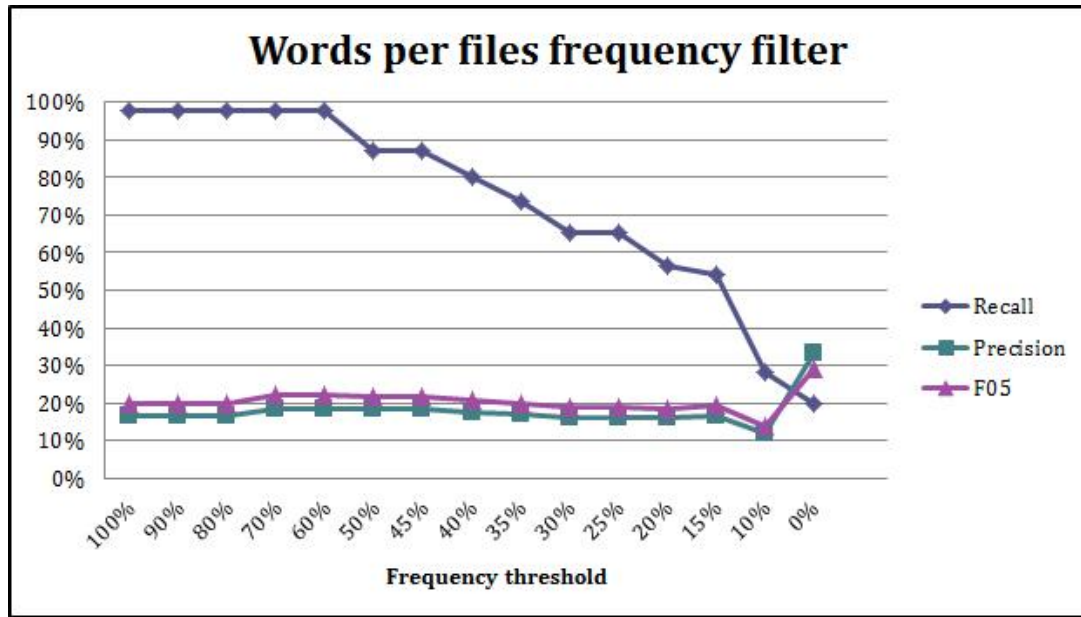


Figure 5.5: Token based approach - Statistical filtering: Filtering out frequent words per file and per language

5.7.4 Weight filtering

Description

Another observation made after the examination of the results, was that many False Positives were caused by dependencies in which only a small number of words were matched from the two dependent files. Examples of such words are *in*, *of*, *id*, *encoding*, *type*, etc. In order to remove these FP the concept of “*weight*” is introduced, as a property of a dependency. The value of the weight is the number of the words which were matched in the two files that the dependency refers to.

Subsequently, a filter which filters out dependencies with a low weight was created. It gets threshold as an input, i.e., a number, below which every dependency is removed.

Evaluation

Fig. 5.6 shows that the recall drops with a near linear behaviour. This happens because there are, indeed, true dependencies which connect two files with a few words. They are removed with the weight filter, and therefore the recall decreases.

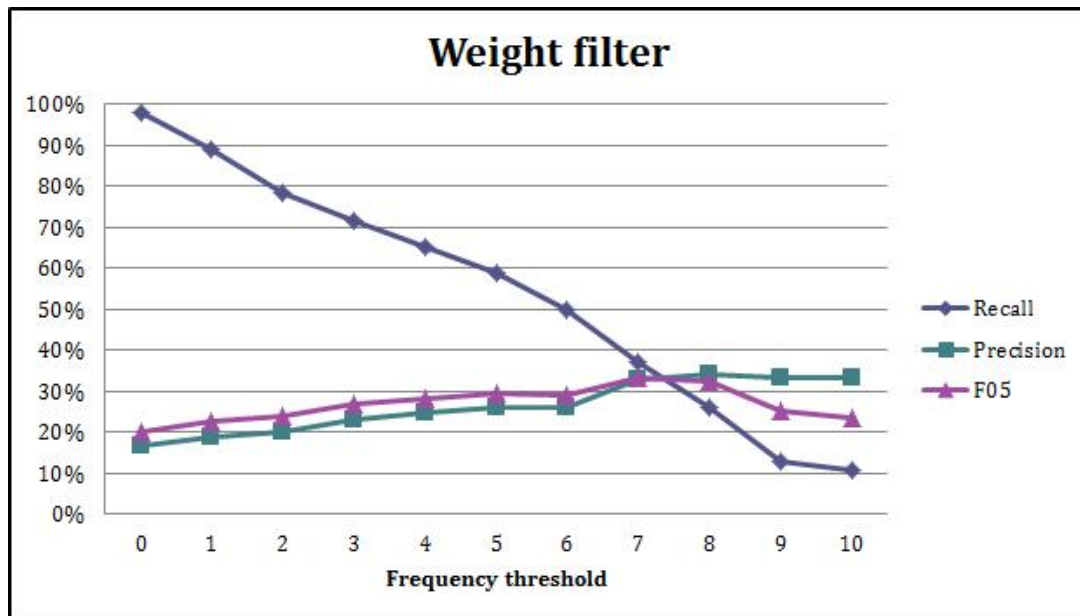


Figure 5.6: Token based approach - Statistical filtering: Weight filtering

5.7.5 Composite experiment

Once we know the behaviour of each filter, the last three filters were applied simultaneously with all the combinations of thresholds that were presented in the previous section. The first one, i.e., *word frequency filter* was omitted because the *word frequency per language filter* is better, and also covers the cases of the former. The results contain combinations of filters and their thresholds, which were better in terms of the accuracy metrics, than the ones obtained by applying the filters individually. Table 5.11 displays the best results for recall, precision and $F_{0.5}$.

5.7.6 Comparison between statistical and ad-hoc approaches

At this point, the experiments with both the ad-hoc and the statistical filters are completed, therefore the two approaches can be compared. The best results from the ad-hoc approach were achieved with the experiment with id=13, using all the filters which were mentioned in the ad-hoc filtering section (see Table 5.12).

We observe that the accuracy of the ad-hoc solution is slightly better than the accuracy of the statistical one, yet the explainability and the extensibility of the statistical solution is by far greater because the basic algorithm is simple and the filters are apply in the same manner to any type of source file. In order to handle a new language with the statistical approach, only the the infrastructure need to be extended, but nothing specific

Table 5.11: Token based approach - Statistical filtering: Composite experiment

Id	Recall	Precision	F_{0.5}	Parameters
665	98%	21.7%	25.7%	Freq. threshold: 100%, Freq. per file threshold: 75%, Min. weight threshold: 0
655	74%	37.4%	41.5%	Freq. threshold: 100%, Freq. per file threshold: 85%, Min. weight threshold: 2
690	68%	41.5%	45%	Freq. threshold: 100%, Freq. per file threshold: 55%, Min. weight threshold: 1
691	60%	50%	51.7%	Freq. threshold: 100%, Freq. per file threshold: 55%, Min. weight threshold: 2
715	40%	62.5%	56.2%	Freq. threshold: 100%, Freq. per file threshold: 35%, Min. weight threshold: 2
726	36%	66.7%	57%	Freq. threshold: 100%, Freq. per file threshold: 25%, Min. weight threshold: 1
716	28%	77.8%	57.4%	Freq. threshold: 100%, Freq. per file threshold: 35%, Min. weight threshold: 3

Table 5.12: Token based approach - Ad-hoc filtering: Best result

Id	Recall	Precision	F_{0.5}	Description
13	97.8%	25.3%	29.7%	filtering out JSP EL, DTD & DOCTYPE words

needs to be done for the method itself.

However, the parameters for the filters need to be determined, i.e., the threshold values, which provide the best accuracy results. An advantage of this parametrisation is that we can focus on a different aspect of accuracy for the analysis. We can focus on the recall if we are interested not to have FN, or focus on the precision and the $F_{0.5}$ if we want to eliminate the FP.

5.8 Ground truth data (*GTD*) amendment

After studying the results the following observations were made:

- A noticeable percentage of the FP were dependencies which were not entirely false, but revealed a weaker relationship between two files. We remind the reader that with the ground truth method we searched only for strong dependencies that is, *direct* and *framework* dependencies.
- We did not search for any dependency between the different XML technologies, e.g., how the presentation logic configuration (JSF configuration) is bonded with the application framework (Spring configuration), or the latter with the database configuration (JPA configuration). These configurations use XML files which were treated as one language.

The aforementioned observations were discussed with consultants and researchers from SIG, in the form of informal interviews and the conclusion was that the ground truth method and data, should be amended, as it is explained in the following paragraphs.

5.8.1 Adding *Indirect* dependencies

As stated in the classification section, *indirect* are dependencies which are weaker than *direct* or *framework* because they may not cause runtime errors, but there is a high probability that a change in one of the dependent files, will require a change in the other file in order to achieve the desired functionality.

In our case study an example is the booking form of a hotel (in JSF), and the database entry for the hotel (in SQL). The field “hotelname” in the JSF is dependent on the database column “hotelname” as it shows its content. If the column is changed to “hoteladdress” the content in the JSF will be wrong.

For that reason all the the files in the case study project were again examined. The *indirect* dependencies that were found were 34, thus increasing the number of total dependencies to 86. The results are shown in Table 5.13 under the first amendments column.

In the same column we can see that eight *framework* dependencies were added. These were discovered during examining the FP for the aforementioned experiments. The eight dependencies were not FP but true dependencies which we missed with the manual checking. This also shows the usefulness of our method.

5.8.2 Treating different XML based technologies as different languages

As stated above, the fact that XML technologies are treated as one language, hides a part of the complexity of a system. Different technologies can belong to different external components or libraries and this is a fact that the software evaluator should be aware of, when judging the complexity of a system.

In examining our case study project the following different technologies (configuration types) were found:

- WebappXML - the configuration for Java web applications
- SpringXML - the configuration for the Spring Framework
- JSFXML - the configuration for Java Server Faces presentation layer
- DBXML - the configuration for Database Access layer

The examination of the case study project revealed eight dependencies among the XML files, one *direct*, six *framework* and one *indirect*. The modified ground truth data are shown in Table 5.13 under the second amendments column. The necessary modifications were made in our tool to accommodate the above changes.

Table 5.13: Amended ground truth data

Attributes	Phases		
	Initial	1st amend.	2nd amend.
Direct dependencies	5	5	6
Framework dependencies	39	47	53
Indirect dependencies	0	34	35
Total relevant dependencies (π)	44	86	94
Collection (C)	367	367	400
Non relevant dependencies (<i>Total negatives</i>)	323	281	306

5.9 Filtering out frequent words per file and language category

5.9.1 Description

Here the concept of language category is introduced, i.e., languages with similar structure, syntax and grammar are grouped. In our case, all XML technologies that are mentioned in the previous paragraph belong to the same category.

Another version of the *Words per files frequency filter* was created, which uses the same formula (see eq.5.4) but the files are counted per language category, and not per language:

$$\begin{aligned} \text{Word freq. per files} &= \\ &= \frac{\text{Number of files from a language category that contain the word}}{\text{Total number of files from this language category}} \end{aligned}$$

As the behaviour of the filter is already known, it was not applied separately, but in combination with the filters of the composite experiment as shown in the next section.

5.9.2 Composite experiment with amended *GTD*

Description

Like in the previous composite experiment, the three following filters with a wide range of thresholds were simultaneously applied:

- *Words per files per language filter*
- *Words per files per language category filter*
- *Weight filter*

The *Words frequency per language filter* was skipped because, from the previous composite experiment, the best results were taken when the threshold for it was 100% thus, it did not remove any words.

Evaluation

From all the threshold combinations, the most interesting ones are displayed in Table 5.14, regarding the score for the three different accuracy metrics. The same experiment results are displayed in Fig. 5.7 which plots the recall on the Y_{axis} and the precision on the X_{axis} .

Table 5.14: Composite experiment with amended *GTD* results

Id	Recall	Precision	F_{0.5}	Parameters
3606	94.7%	37.9%	43.1%	Freq. word/files/language threshold:60%, Freq. word/files/category threshold:55%, Min. weight threshold:0
3620	92.6%	38.5%	43.6%	Freq. word/files/language threshold:60%, Freq. word/files/category threshold:50%, Min. weight threshold:0
3688	87.2%	43.9%	48.7%	Freq. word/files/language threshold:60%, Freq. word/files/category threshold:30%, Min. weight threshold:0
3705	85.1%	48.2%	52.8%	Freq. word/files/language threshold:60%, Freq. word/files/category threshold:25%, Min. weight threshold:0
4706	39.4%	80.4%	66.6%	Freq. word/files/language threshold:70%, Freq. word/files/category threshold:30%, Min. weight threshold:4
4730	16%	93.8%	47.6%	Freq. word/files/language threshold:35%, Freq. word/files/category threshold:25%, Min. weight threshold:4

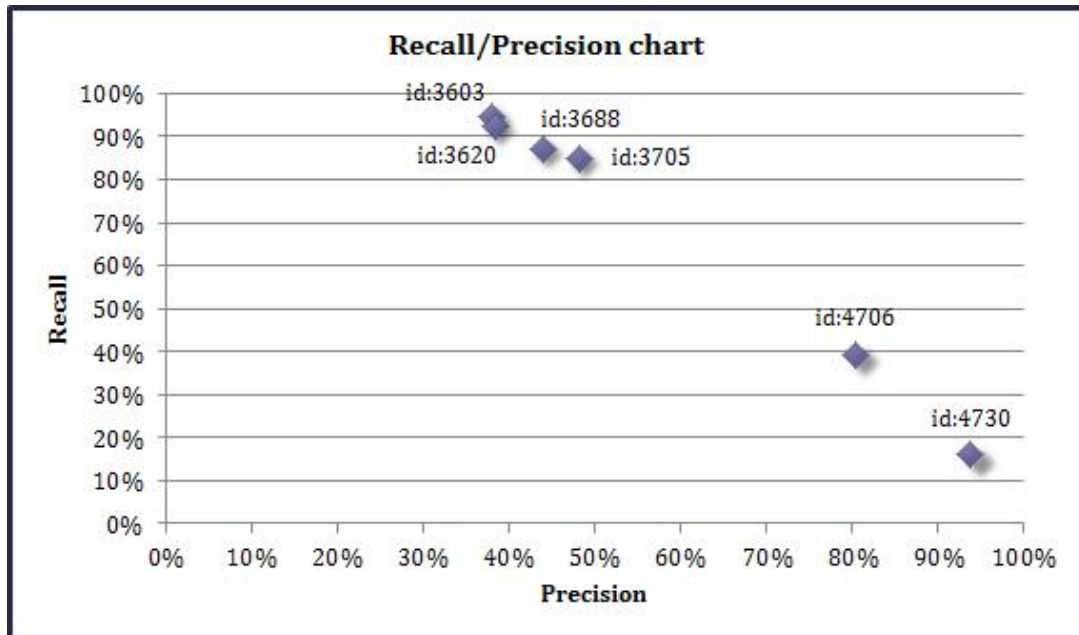


Figure 5.7: The chart plots the most interesting results of the second composite experiment after the amended ground truth data. The recall is plotted on the Y_{axis} and the precision on the X_{axis} .

As can be seen from the table and the chart, it is difficult to achieve a parametrisation which results in very good values for all three metrics (i.e., recall, precision and $F_{0.5}$). However depending on the desired priority of the accuracy aspect, a high value in the corresponding metric can be achieved, with a trade-off in another aspect. In our case, nearly 95% of the dependencies can be captured, with a precision of 38%. This means that for each true dependency that is captured, 1.6 false dependencies are captured by mistake. On the other hand, if precision is the top priority, an 80% precision can be achieved while capturing 40% of the dependencies. The conclusion is that it is possible to obtain a desired combination of values with a simple words matching approach.

Chapter 5. Proposed method

Chapter 6

Large scale case study project

6.1 Project description

The case study project on which all the experiments were conducted was very small (around 3,200 LOCs). Therefore it would be questionable to generalise the applicability of the method configuration and its results. That is why the proposed method was executed on a large scale project from the company. The specific project was selected for the following reasons:

- We are familiar with the involved technologies e.g., Java, Spring Framework, Web Services (WSDL files).
- The project was based on Web Services, which is a dependency type that the consultants were interested in.
- The project was still under evaluation by the company; therefore the consultants assigned to it could be contacted for further information.

An analysis of the content of the project revealed that it contained about 15,000 files distributed mainly in Java, XML, WSDL and JSP files. The project also contained a few files written in HTML, JavaScript and SQL. The languages that had the highest probability to participate in a web services related dependency in the specific project were Java, WSDL and XML. As our goal was to extract the web services dependencies, the rest of the languages (JSP, HTML, JavaScript and SQL) were excluded from the analysis.

The project is based on the Spring Framework¹ which is a Java based enterprise application framework.

¹<http://www.springsource.org/spring-framework/>

6.2 Extension for another language of the project

The tooling did not support the WSDL language, therefore the necessary changes had to be implemented to the prototype in order to make our method applicable for WSDL. This was a good opportunity to evaluate the extensibility of our method. Every change performed, and the total time spent were documented. Due to the confidentiality of the SIG's underlying infrastructure the main changes are briefly mentioned:

- The process of accepting files with extension “.wsdl” with a new entry in the FileFilters component.
- A new entry in the supported languages set.
- A new entry in the XML category of languages (see section 5.9) because WSDL is an XML type of language.
- An assignment of the WSDL language to the general token-based pattern.
- Unit tests for every new entry.

The fact that WSDL is XML based is coincidental. It would take the same time, if there was a different type of language at hand. The whole implementation and testing process lasted less than three hours, because only infrastructure changes were needed. It would have been better if another developer extended the method, because this would increase the objectiveness of the extension time. However the time of the developers was limited and implementing these changes ourselves was the only solution. From the reviewed state of the art methods, it is assumed that none could be extended so fast because, apart from the infrastructure work, parsers or modelers need to be created which is a time consuming process. Therefore, our proposed method has an advantage in extensibility.

6.3 Challenges

The biggest challenge that was encountered with the large scale project, was performance. It took around seven hours to complete an analysis on a desktop computer of moderate capabilities (4x2.8Ghz processor and 3GB memory). Because the time was limited and it was needed for experimentation and analysis, many parts of our implementation were optimised such as the application of the filters and the token based matching. With these optimisations the computation time decreased to half, while ensuring that the algorithm results were not affected.

The duration was still long, but the time frame of the thesis did not allow us to consider further improvements. Therefore, the suggestions of the consultants of the project were followed, to apply the method only to the Java and WSDL technologies. When the language set was reduced to these two languages, the project could be analysed in around 20 minutes (the project contained around 8,000 Java files and 400 WSDL files).

6.4 Evaluation method for large scale systems

The manual checking of dependencies was limited in this project because the number of files is large and consequently the number of retrieved dependencies is large as well. The dependencies of the project could not be extracted with the ground truth method as it would take, with an optimistic estimation, more than three weeks to finish. Therefore the recall could not be calculated. What we did was:

1. Take a random sample of the retrieved dependencies. All the dependencies in the sample should be different. This sampling technique is called “Random sampling without replacement” (Barreiro and Albandoz, 2001).
2. Check their validity manually.
3. Calculate the precision for this sample.
4. Generalise the precision result to the whole project.

In the next experiments the evaluation is based on the above steps.

6.5 Experiment

In order to test the generalisability of the threshold settings, a setting that was applied on the small case study was used, which provided high precision results (80.4%), with moderate recall (39.4%). This setting was selected because the precision was more important for the consultants than the recall.

The method retrieved around 10.000 dependencies. By examining the results we realised that a big part of the dependencies were created because of words that contained two to four letters. Our hypothesis was that the dependencies with words containing a few characters are more probable to be False Positives. A sample of the dependencies was taken where the matched words consisted of less than five characters. Our hypothesis was confirmed because, in this set, the precision was 5%.

6.6 Refinement

The algorithm was refined to capture only words with five, or more characters. The rationale is that in a big project, the class, method names (for Java) or the entities in general that cooperate with other components usually have longer names in order to carry semantics which are useful for the maintainability of the source code.

The new analysis retrieved 630 dependencies. By examining a random sample of them, the precision was found to be improved but it was still low (23.5%). In order to explain this score, a histogram of the files which comprised these dependencies was created (see

Fig. 6.1). From the 8,500 WSDL and Java files, 266 different files participated in our dependencies set. By analysing the histogram and the results we found that from these 266 files, six files only were responsible for 27% of dependencies.

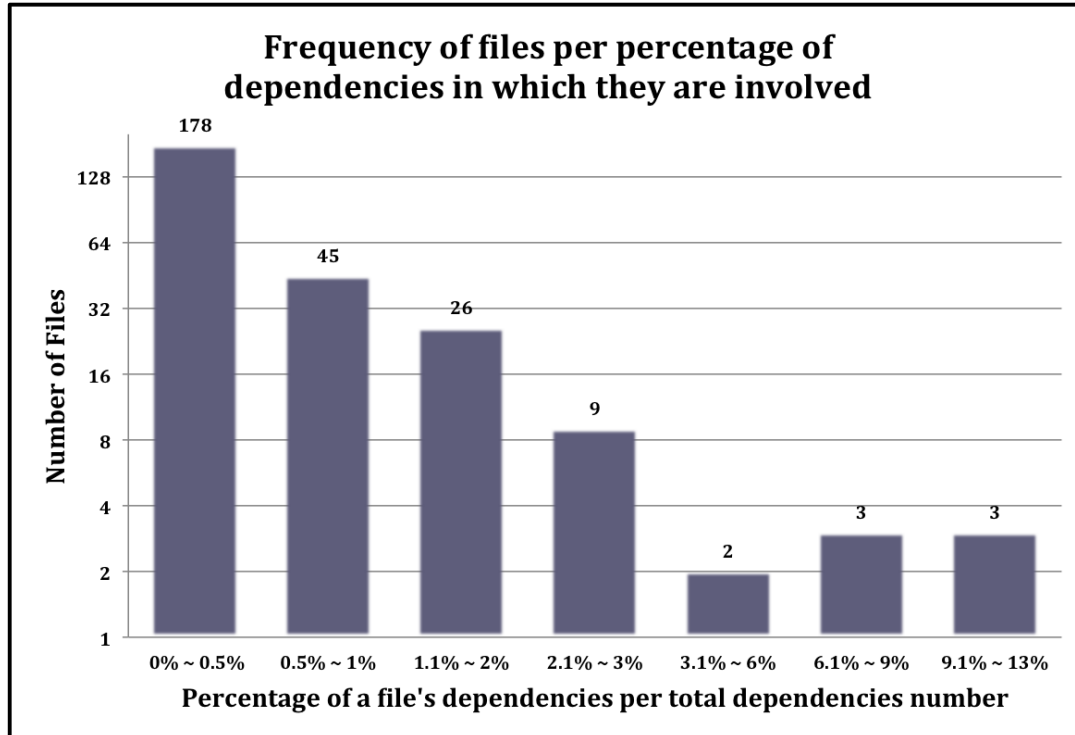


Figure 6.1: The histogram plots the number of files per percentage of dependencies in which they are involved.

From the six files, one was a Java interface, and one was a Java abstract class. They were both implemented/extended by most of the other Java files which participated in the Web Services interfacing. They declared methods described in the WSDL files, therefore they were true dependencies. The other four files were WSDL files which had very long free text within the documentation tags describing the methods. This turned out to be the source source of many False dependencies. More WSDL files were examined and we observed that they also contained long text within their documentation tags.

The algorithm was refined again, to exclude the documentation tags, as this case was clearly project specific. The analysis was run again and the number of dependencies reduced to 157. With the same sampling technique, a sample of 30 files was taken and the precision was 76.7% which was rather satisfactory.

6.7 Evaluation

The filtering of the words with less than five characters is generic and it can be applied to any project, with perhaps a configurable value for the number of characters. However the exclusion of the documentation tags from the WSDL files was a project specific refinement of the method. Therefore more projects to test its generalisability are needed. Data may not be available to calculate recall, but it was proven that the method was able to support a new language in less than a day of development, and to provide results with high precision (76.7%).

Chapter 7

Discussion

In this chapter the results of the thesis, the problems that were encountered, controversial points and possibilities for future research are discussed.

7.1 Answers to research questions

As stated in section 1.3, *the objective of this study is to improve the process of detecting dependencies across programming languages, focusing on the cost-effectiveness and the accuracy, from the viewpoint of a software evaluator, in the context of evaluating heterogeneous information systems.* We believe that we managed to reach our objective, because all our research questions were answered, through the different phases of the research, as shown in Table 7.1.

In Table 7.2 the proposed method is compared with the state of the art methods for the MoSCoW's *Must have* requirements. The rest of the requirements were not included because we could only focus on the most important ones, given the time limitations of this thesis.

Our method can be explained to general consultants, can be extended for a new language in less than a day and we managed to have an accuracy level with $F_{0.5}=66.6\%$. We believe that the answers to the sub-questions, and the result of this comparison, answers the main research question as well, which is:

RQ: What is a cost-effective, yet accurate method of detecting the dependencies across software modules, in heterogeneous software systems?

Table 7.1: Answers to research (sub)questions

Research sub-question	Answer
<i>SQ₁: What is a dependency?</i>	Our first sub-question was answered by literature study in section 3.1.1.
<i>SQ₂: What types of dependencies exist across languages?</i>	<i>SQ₂</i> was answered by literature study in section 3.1.2 where the dependencies are classified according to their strength and their functionality.
<i>SQ₃: How can we detect these dependencies?</i>	In the beginning, the state of the art methods were discovered and evaluated. Then we realised that the existing methods focus on the accuracy, whereas we focus on the balance between cost-effectiveness and accuracy. In chapter 5, a new method was proposed which uses a simple algorithm that minimises the input needed for analysis, thus making it extendible and explainable.
<i>SQ₄: How can we evaluate the state of the art?</i>	A literature study was conducted in order to discover the state of the art methods in section 3.2. Then with literature study and expert interviews (see section 2.3) the requirements that these method should abide to were defined. A rating criteria schema was created in section 4.2 with which the reviewed methods were evaluated. The result is the comparison matrix in section 4.7.
<i>SQ₅: How can we evaluate the accuracy of the method?</i>	In section 4.3 the metrics used for accuracy were defined in detail by studying the literature. A ground truth method was used to manually extract the dependencies from a small case study project in section 2.5.1, and we compared with them the results that the proposed method produces. The proposed method was executed and evaluated on a large scale project too.
<i>SQ₆: How can we evaluate the cost-effectiveness of the method?</i>	The extensibility and explainability of the method are key elements of its cost-effectiveness. These two requirements and their rating criteria were defined in section 4.2 and the time needed for the method to be implemented for an unsupported language was measured in section 6.2.

Table 7.2: Comparison matrix of the state of the art methods and the proposed method according to their rating on the MoSCoW’s Must have requirements.

Requirement	Eclipse MoDisco	PAMOMO	GenDeMoG	Moose	DSketch	Proposed method
Accuracy	?	?	?	?	?	****
Explainability	?	**	***	***	***	****
Extensibility	**	**	**	***	*	*****

7.2 Validity & Reliability

Yin (2009) introduces the measurements of validity and reliability for assessing how trustworthy the research results are and how accurately they reflect the reality. He divides validity into three categories:

- **Construct validity:** The extent to which the construct measures the characteristics being investigated.
- **Internal validity:** The extent to which the results are accurate and conforming to reality.
- **External validity:** The extent to which the results of the research can be generalised.

Reliability demonstrates that the operations of our study can be repeated with the same results.

The next paragraphs describe the elements of the thesis that increase and are also a threat to validity and reliability.

Construct validity

Construct validity is achieved by using multiple sources of evidence, establishing a chain of evidence, and by letting the thesis artefacts and methods be review by key informants. The elements of the thesis that increase the construct validity are:

- Throughout the thesis document, citations were made to all the sources of information that have been used.
- The set of requirements was created using a literature study, informal interviews, and it was validated by three business and three technical consultants with at least two years in software evaluation.
- The results from every experiment were stored in a database or in text files for analysis and validation. Apart from the values to the accuracy metrics, they contain analytical information about the False Positives, the False Negatives, the filtered words (by the frequency filters) and the filtered dependencies (by the weight filter) so that they are available for checking at any time during the research.
- A software tool was implemented to compare the method results with the ground truth data in a replicable way.
- This thesis was supervised by four supervisors, two from the university and two from SIG.

And the possible threats to construct validity are:

- The interviews were conducted with only six subjects, all from the same company (SIG).

- In the second phase of the interviews, after some time to discover the requirements, the interviewees were presented with a predefined (by us) set of requirements. If more time had been available, workshops for discovering and prioritising the requirements would be a better, more unbiased solution.

Internal validity

Internal validity refers to the establishment of causal relationships and explanations between data, variables, methods and results. In this thesis internal validity is increased due to the following reasons:

- The first three phases of the interviews contained exactly the same questions, and were conducted in the same manner for every participant. Only the last part was explorative, where we tried to get any useful information for our research.
- Manual checking was selected as a method to extract ground truth data. For dependencies that involved uncertainty, triangulation was achieved, using the advice from software evaluators at SIG.
- For every algorithm that was selected for the proposed method, the rationale behind it was documented. The refinements to the algorithms were selected after analysis of the results, i.e., False Positives, False Negatives, filtered words and dependencies.

Serious threats to internal validity were not identified.

External validity

External validity is concerned with the degree of generalisation of the study's findings beyond the case studies.

We tried to generalise our findings by executing the method on a large scale project, using the parametrisation that produced the best results from the small case study.

However, external validity is not the strongest part of this thesis because of the following threats:

- Recall could not be measured in the large scale case study, because extracting the actual dependencies was practically impossible.
- The same parametrisation of the method gave similar, but not the same results in the small and the large scale projects (the difference in precision was 3.7%). A custom refinement had to be done to the large scale project to achieve this result.
- Two case studies are indicative for our results, but are not considered enough to generalise our results.

Reliability

Reliability is concerned with the degree to which the method and the findings of this research can be reproduced by other scientists.

- The rating schema has specific metrics, which can be used to the same state of the art methods and produce the same results.
- The algorithm of the method is documented analytically, and can be executed in a reproducible way.
- The small case study involved a small open-source project with common modern technologies so that the manual checking and the execution of the method can be performed in a repeatable manner. The results of the manual checking are available for re-use in the Appendix B.

Threats to reliability:

- The large case study is not repeatable because the source is not publicly available.
- The extensibility experiment is based on the SIG framework, therefore it cannot be reproduced. An underlying framework has to be constructed by future researchers.
- The source code of the implementation cannot be published because it depends on the SIG infrastructure.

The aforementioned elements that increase and threaten the validity and the reliability of this thesis are displayed in Table 7.3.

Table 7.3: A summary of the four measurements according to Yin (2009), which assess the validity and the reliability of this thesis. The elements that have positive and negative effect on each measurement are in the corresponding columns of the table.

Measurem	Positive	Negative
Construct validity	<ul style="list-style-type: none"> -Citations to all the literature sources. -Requirements were created with a literature study, informal interviews, and validated by experts. -Experiment results stored in detail into database or files. -Implementation of a software tool for validating the method. -Thesis supervised by four supervisors. 	<ul style="list-style-type: none"> -The interviews with only six subjects, all from SIG. -Interviews instead of workshops to define requirements
Internal validity	<ul style="list-style-type: none"> -Structured type interviews (the first three phases). -Ground truth data by manually extracting dependencies. Triangulation with experts in case of uncertainty. Unit tests for testing the tool. Automated comparison of the method's results with the ground truth data. -Documentation of the rationale for every algorithm. Refinements to the algorithm after systematic analysis of the results. 	
External validity	<ul style="list-style-type: none"> -Case study on a large scale project, using the knowledge from the small case study. 	<ul style="list-style-type: none"> -Recall not measured in the large scale project because manual extracting of dependencies was impractical. -The same parametrisation of the method gave similar, but not the same results in the small and the large scale projects. -Custom refinement to the method for the large scale project was necessary. -Two case studies are indicative of the method's behaviour, but generalisation is under discussion.
Reliability	<ul style="list-style-type: none"> -Rating schema with specific metrics. -Analytical documentation of the algorithm. -Open-source project for the case study, with modern technologies. The results of the manual checking are available in Appendix B. 	<ul style="list-style-type: none"> -Large case study is not repeatable due to closed source code. -The extensibility experiment is not reproducible. -Source code of method implementation cannot be published.

7.3 Future research

An important addition to this thesis would be a benchmark of the method behaviour for categories of projects. For that, we need a classification of a set of projects according to their language set and their size. Then the proposed method can be executed on every project with a wide range of parameters (different combinations of threshold values to the statistical filters). For every analysis the results should be checked and compared with the analyses from the other projects in the same category. If the behaviour is similar, then we can conclude on the best parametrisation for each category of projects. After that, the input for the method can be extracted from the results of the benchmark, which is a process that can be automated.

In addition, an observation that was made during all the experiments, was that the cross-language dependencies were extracted using words that contained, or were similar to nouns and verbs (e.g., *Booking*, *getBooking* and *bookHotel*). Therefore future research can be conducted on the use of physical language dictionaries and the use of similarity algorithms in order to restrict the extraction of words from the source files.

Although there is nothing in the algorithm to prevent the method to be executed for same-language dependency analysis, only cross-language experiments were conducted by excluding the same-language findings. Therefore we can only have an assumption about the method's behaviour. An experimentation with the same way and the same evaluation process as discussed in this thesis can be conducted for to verify this assumption.

Research can also be conducted on the behaviour of the method at component level. In order to do that, the file level dependencies need to be aggregated into component level ones. We may be able to increase the accuracy of the method by applying other type of filters on the components or their dependencies. These types of filters can also be the objective of future research.

Moreover, another interesting research possibility, would be to combine the token based statistical approach with the pattern based. For example, if just the table names of the SQL files are interesting for the analysis, these can be extracted with a pattern based approach, which is language specific. Then these table names can be matched with word nodes that are extracted from files from other languages with the token based approach. Statistical filtering can still exclude frequent words, without being applied to the SQL files. Therefore, with the combination of minimal language specific information, and the token based approach, there is a probability for increasing accuracy.

Chapter 7. Discussion

Chapter 8

Conclusion

In this thesis we introduced an innovative method for detecting dependencies across software modules which are developed in diverse programming languages (i.e., in heterogeneous systems). The accuracy of the method was found to be satisfactory ($F_{0.5} = 66.6\%$). The method's greatest advantages are:

- the explainability due to its simple algorithm and
- the extensibility due to the fact that the algorithm is so generic that can be applied on any language without additional input to the method.

These are properties that make the method highly cost-effective, thus appealing, to be used by software evaluators, software maintainers, technical and general consultants.

This flexibility of our method is achieved by the fact that it does not use language or project specific information or any other ad-hoc approach to detect dependencies. On the contrary, the base idea is that with a simple token based algorithm tokens/words from the source files are extracted, and then statistical filtering, based on statistical attributes of these words, is applied. After unwanted words are filtered out, the files that have the same words in common are marked as dependent. An example of a statistical filter is the one that removes the words that appear in many different files of the same language. This filter would remove words such as *public*, *class*, *private*, etc. within the Java files.

There are two different aspects of accuracy in which we are interested: *recall*, which is the percentage of actual dependencies that are retrieved out of the whole actual dependencies of a project, and *precision*, which is the percentage of actual dependencies among the retrieved dependencies. Depending on the parametrisation of our method, namely of the statistical filters, the two aspects can have different values (e.g., around 85% recall with a 48% precision, or 80% precision with around 40% recall). These levels of accuracy are satisfactory for our purpose because according to the consultants, precision is more important than recall. However there are possibilities to improve accuracy. In section 7.3, ways in which future research can increase accuracy are presented. e.g., by adding new statistical filters such as the one that excludes words with a few characters and by creating benchmarks which will provide the best parametrisation for projects with similar languages set and size.

Furthermore, a contribution of this work is the detailed description of the process that was executed to evaluate our method's results. The evaluation method states our dilemmas and the rationale for our decisions for the selection of the case study projects and the ground truth method which was used for comparison. It is described in such a way that it is reusable for future research in this domain.

Last but not least, a comparison between the state of the art methods that were found in the literature is presented. In order to do that, the most important requirements for a dependency detection method were defined. A five level Likert's type scale with a description for each level was used to evaluate the methods against these requirements. We showed with this comparison that there are other perspectives apart from accuracy which are also very important for the software evaluators, i.e., explainability and extensibility.

We consider that our initial objective was reached, which was to find a cost-effective, yet accurate method for detecting dependencies in heterogeneous systems. We believe that this thesis has an actual contribution both to the academic and to the business environment and also that it has created the foundations for future research.

Bibliography

- Ahl, V. (2005). An experimental comparison of five prioritization methods. *Engineering*, (August).
- Alvarez, S. (2002). An exact analytical relation among recall, precision, and classification accuracy in information retrieval. *Boston College, Boston, Technical Report BCCS-02-01*, pages 1–22.
- Arnold, W. E., McCroskey, J. C., and Prichard, S. V. O. (1967). The likert-type scale. *New York*, 15(2):31–33.
- Bagchi, S., Kar, G., and Hellerstein, J. (2001). Dependency analysis in distributed systems using fault injection : Application to problem determination in an e-commerce environment. *October*, pages 1–13.
- Barreiro, P. L. and Albandoz, J. P. (2001). Population and sample . sampling techniques. Technical report, University of Seville, Management Mathematics for European Schools.
- Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). The goal question metric approach. *Science*, 2:1–10.
- Berander, P. and Andrews, A. (2005). *Requirements Prioritization*, pages 69–94. Number November. Springer-Verlag.
- Beyer, D. and Noack, A. (2005). Clustering software artifacts based on frequent common changes. *13th International Workshop on Program Comprehension IWPC05*, pages 259–268.
- Bruneliere, H., Cabot, J., and Jouault, F. (2010). Modisco : A generic and extensible framework for model driven reverse engineering. *IEEEACM International Conference on Automated Software Engineering*, (Ase):173–174.
- Cossette, B. and Walker, R. J. (2010). Dsketch : Lightweight , adaptable dependency analysis. *Change*, pages 297–306.
- de Souza, C. (2005). *On the relationship between software dependencies and coordination: field studies and tool support*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE.
- Ducasse, S., Girba, T., Kuhn, A., and Renggli, L. (2009). Meta-environment and exe-

Bibliography

- cutable meta-language using smalltalk: an experience report. *Software and Systems Modeling*, 8(1):5–19.
- Guerra, E. and de Lara, J. (2010). Pamomo. <http://astreo.ii.uam.es/eguerra/tools/pamomo/main.htm>. Retrieved December 10, 2012.
- Guerra, E., de Lara, J., Kolovos, D., and Paige, R. (2010). Inter-modelling: From theory to practice. In Petriu, D., Rouquette, N., and Haugen, ., editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, pages 376–391. Springer Berlin / Heidelberg.
- Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems. *MIS Quarterly*, 28(1):1–7.
- ISO25010 (2011). ISO/IEC 25010: 2011, Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-System and software quality models.
- Jansen, S., Brinkkemper, S., and Hunink, I. (2008). Pragmatic and opportunistic reuse in innovative start-up companies. *Software, IEEE*, pages 42–49.
- Jouault, F., Vanhooff, B., Bruneliere, H., Doux, G., Berbers, Y., and Bzivin, J. (2010). *Inter-DSL coordination support by combining megamodeling and model weaving*, pages 2011–2018. ACM Press.
- Knudsen, L. G. (2004). Developing an OCT imaging system in java. Master’s thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU.
- Kvale, S. and Brinkmann, S. (2009). *InterViews: Learning the Craft of Qualitative Research Interviewing*, volume 20. Sage Publications.
- Levy, Y. and Ellis, T. J. (2006). A systems approach to conduct an effective literature review in support of information systems research. *Science Journal*, 9(1):181–212.
- Manning, C., Raghavan, P., and Schuetze, H. (2009). *Evaluation in information retrieval*. Number c. Cambridge UP, online edi edition.
- McCabe, T. J. (1976). Ieee transactions on software engineering. *A complexity measure*, (2(6)):308–320.
- Moise, D. and Wong, K. (2005). Extracting and representing cross-language dependencies in diverse software systems. *12th Working Conference on Reverse Engineering (WCRE05)*, pages 209–218.
- Nagappan, N. and Ball, T. (2007). Using software dependencies and churn metrics to predict field failures: An empirical case study. *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 364–373.

Bibliography

- Pfeiffer, R.-H. and Wasowski, A. (2011). Taming the confusion of languages. In *Proceedings of the 7th European conference on Modelling foundations and applications, ECMFA'11*, pages 312–328, Berlin, Heidelberg. Springer-Verlag.
- Spanoudakis, G. (2005). Software traceability: a roadmap. *of Software Engineering and Knowledge*, III:1–35.
- Van Den Brink, H., Van Der Leek, R., and Visser, J. (2007a). Quality assessment for embedded sql. *International Working Conference on Source Code Analysis and Manipulation SCAM*, pages 163–170.
- Van Den Brink, H., Van Der Leek, R., and Visser, J. (2007b). Quality assessment for embedded sql. *International Working Conference on Source Code Analysis and Manipulation SCAM*, pages 163–170.
- Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36.
- Van Rijsbergen, C. J. (1979). *Information Retrieval*, volume 30. Butterworths.
- Webster, J. and Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2):xiii–xxiii.
- Wilde, N. (1990). Understanding program dependencies. *Program*, (SEI-CM-26).
- Yin, R. K. (2009). *Case Study Research: Design and Methods*, volume 5. Sage Publications.

Appendix

Appendix A - Interview form for defining requirements

Interview

Interviewee's name:

Date:

Type of consultant:

Years of experience in evaluation:

The purpose of this research is to develop a method, namely an algorithm, with which we can detect the dependencies across software modules that are developed in diverse programming languages, e.g. systems that contain components in Java, C/C++, Databases, XML files etc. The method has to be developed, applied and evaluated in 3 months.

With this interview we are trying to define the requirements (RQs) of that method.

Phase 1

Q1. *In your opinion, what should be the 5 most important RQs of a method for detecting dependencies across software modules that are developed in diverse programming languages? Can you explain why?*

Table : Interviewee's RQs

Requirement	Description/Explanation

Phase 2

We believe that the following set of RQs is related to the method. Please read them and ask if something is unclear to you.

Table : Predefined RQs set

Requirement	Description
Extensibility	The method should be able to be extended so as to include an unsupported language in a short time.
Accuracy	The percentage of false positives and false negatives should be as low as possible.
Initial range of languages	The method initially should support a set of different programming languages.
Explainability	The meaning of the dependency findings should be able to be understood by the software evaluator.
Efficiency	It should be fast enough so that it can be applied to large-scale systems in a reasonable timeframe.
Exportability	The results should be exportable to open format(s).
Visualization	The results should be displayed in a visual way.
Filtering	The results can be filtered (e.g. according to keywords or dependency types).
Tool integration	The method should be able to be implemented as part of an existing tool.

Q3. *Could you please order the RQs in each column, so that the top priority from each column is number 1?*

Q4. *Why did you do that selection and ordering?*

Phase 3

Q5. *Could you please propose a value for each requirement that you think it is appropriate for our method and the specific timeframe we have?*

Table : RQs proposed values

Requirement	Indicative Value Type	Value
Extensibility	In man-days	
Accuracy	AVG (FP, FN)	
Initial range of languages	Number of languages	
Explainability	N/A	
Efficiency	Time for processing 10 KLOC	
Exportability	File formats	
Visualization	Visualization type	
Filtering	Filtering criteria	
IDE integration	Tool name	

Phase 4

Q6. *Could you name projects from this company in which you know from your experience that the systems which the projects involved, contained interdependent components developed in diverse languages?*

Q7. *Do you remember what types of dependencies you encountered among those components?*

Appendix B - Amended *GTD* for small case study project

Booking Faces Dependencies	Amenity.java	Booking.java	BookingService.java	Hotel.java	JpaBookingService.java	ReferenceData.java	SearchCriteria.java	User.java	persistence.xml	import.sql	create.sql	data-access-config.xml	security-config.xml	web-application-config.xml	webflow-config.xml	webmvc-config.xml	main-flow.xml	booking-flow.xml	faces-config-12.xml	faces-config.xml	enterBookingDetails.xhtml	reviewBooking.xhtml	enterSearchCriteria.xhtml	reviewHotel.xhtml	reviewHotels.xhtml	intro.xhtml	standard.xhtml	login.xhtml	logoutSuccess.xhtml	index.html	web.xml	messages.properties	
Amenity.java																																	
Booking.java											DB&PROP:F																				PROP:I	PROP:F	
BookingService.java											PROP&DB:I	CONF&PROP:F																					
Hotel.java											DB&PROP:F																						
JpaBookingService.java								CONF&PROP:F			PROP&DB:I	CONF&PROP:F																					
ReferenceData.java																																	
SearchCriteria.java																																	
User.java											DB&PROP:F																						
persistence.xml	CONF&PROP:D		CONF&PROP:D					CONF&PROP:D				DB&CONF:F																					
import.sql										PROP&DB:I																							
create.sql											PROP&DB:I																						
data-access-config.xml											DB&CONF:F	DB&CONF:D																					
security-config.xml										PROP:I	CONF&PRC	PROP:I																					
web-application-config.xml																																	
webflow-config.xml																																	
webmvc-config.xml											CONF:F																						
main-flow.xml	PROP:F	PROP:F	PROP:F	CONF&PROP:F			PROP:F	PROP:F																									
booking-flow.xml	PROP:F	PROP:F	PROP:F	CONF&PROP:F				PROP:F													FIO&PROP:F	FIO&PROP:F											
faces-config-12.xml																																	
faces-config.xml																																	
enterBookingDetails.xhtml	PROP:F	PROP:I	PROP:F	PROP:I	PROP:F						PROP:I																						FIO:F
reviewBooking.xhtml	PROP:F	PROP:I	PROP:F	PROP:I							PROP:I																						
enterSearchCriteria.xhtml	PROP:F	PROP:I	PROP:F	PROP:I	PROP:F	PROP:F	PROP:F	PROP:F				PROP:I																					
reviewHotel.xhtml		PROP:I	PROP:F	PROP:I																													
reviewHotels.xhtml		PROP:I	PROP:F	PROP:I		PROP:F																											
intro.xhtml											PROP:I		PROP:I	PROP:I	PROP:I	PROP:I	FIO:F																
standard.xhtml							PROP:F						FIO:F				PROP:F	PROP:F															
login.xhtml								PROP:I	PROP:I	PROP:I																							
logoutSuccess.xhtml																																	
index.html																																	
web.xml												CONF&FIO:F	CONF&FIO:D			CONF:I																	FIO:D
messages.properties																																	

COUNT Direct dependencies (D) 6 categories:
 COUNT Framework dependencies (F) 53 DBACCESS (DB) Java springxml webxml
 COUNT Indirect dependencies (I) 35 CONFIGURATION (CONF) PIsql jsfxml properties
 COUNT All dependencies (TP+FN) 94 FILEIO (FIO) dbxml jsf / html
 Not Applicable (grey cells) 224 PROPAGATION OF VARIABLE VALUE (PROP)
 No Deps (TN+FP, or empty white cells divided by two) 306
 All possibilities (N) 400