EPM–RT–2010-05

TRACEABILITY IMPROVEMENT FOR SOFTWARE MINIATURIZATION

Nasir Ali
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Septembre 2010

EPM-RT-2010-05

Traceability Improvement for Software Miniaturization

Nasir Ali
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Septembre 2010

EPM-RT-2010-05
Traceability Improvement for Software Miniaturization
par : Nasir Ali
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Traceability Improvement for Software Miniaturization

by

Nasir Ali

Jury Composed of:

Gabriela Niolescu,	président-rapporteur
Yann-Gaël Guéhéneuc,	directeur de recherche
Giuliano Antoniol,	codirecteur
Yvan Labiche,	examinateur externe

A research proposal submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Département de génie informatique et génie logiciel
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Abstract

On the one hand, software companies try to reach the maximum number of customers, which often translate into integrating more features into their programs, leading to an increase in size, memory footprint, screen complexity, and so on. On the other hand, hand-held devices are now pervasive and their customers ask for programs similar to those they use everyday on their desktop computers. Companies are left with two options, either to develop new software for hand-held devices or perform manual refactoring to port it on hand-held devices, but both options are expensive and laborious. Software miniaturization can aid companies to port their software to hand-held devices. However, traceability is backbone of software miniaturization, without up-to-date traceability links it becomes difficult to recover desired artefacts for miniaturized software. Unfortunately, due to continuous changes, it is a tedious and time-consuming task to keep traceability links up-to-date. Often traceability links become outdated or completely vanish. Several traceability recovery approaches have been developed in the past. Each approach has some benefits and limitations. However, these approaches do not tell which factors can affect traceability recovery process. Our current research proposal is based on the premise that controlling potential quality factors and combining different traceability approaches can improve traceability quality for software miniaturization. In this research proposal, we introduce traceability improvement for software miniaturization (TISM) process. TISM has three sub processes, namely, traceability factor controller (TFC), hybrid traceability (HT), and software miniaturization optimization (SMO). TFC is a semi automatic process, it provides solution for factors, that can affect traceability process. TFC uses a generic format to document trace quality affecting factors. TFC results will help practitioners and researcher to improve their tool, techniques, and approaches. In the HT different traceability, recovery approaches are combined to trace functional and non-functional requirements. HT also works on improving precision and recall with the help of TFC. Finally these links have been used by SMO to identify required artefacts and optimize using scalability, performance, and portability parameters. We will conduct two case studies to aid TISM. The contributions of this research proposal can be summarised as follow: (i) traceability support for software miniaturization and optimization, (ii) a hybrid approach that combines the best of available traceability approaches to trace functional, non-functional requirements, and provides return-on-investment analysis, (iii) traceability quality factor controller that records the quality factors and provide support for avoiding or controlling them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traceability is the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship with one another; for example, degree to which requirements and design of a given software's component match [1]. Requirement traceability has received much attention over the past decade. Requirement is a piece of data that change and evolve with time. The change in requirement causes modification in whole software life cycle. For example, a requirement about changing email protocol can cause to update many other dependent modules in the email system. Thus, it becomes important to keep track of software evolution to maintain the system in long run. Traceability links aid in tracing the project status, risk analysis, verification & validation, and other software development activities. Software systems require constant modifications to meet new and ever changing requirements, increasing software complexities and maintenance cost, unless change accommodations is rigorously taken into account as part of the software development process [2].

Several studies [3] have shown that many IT organizations do not practice requirement traceability and therefore encounter delays, failures, and customer dissatisfaction. IT organizations want to satisfy maximum number of customers to offer same solutions for both desktop computer and hand-held device users. The main differences between desktop computers and hand-held devices are memory size, storage size, display size, and computational power. However, it is not cost effective solution to create two versions of the same software for desktop users and hand-held device users. Software miniaturization can aid IT organizations to port their software to hand-held devices. The software miniaturization can be defined as:

1. *Software miniaturization is a form of software refactoring focused on reducing an application to the bare bone [4].*
2. *Software miniaturization can be thought of as software refactoring where the focus is on the optimization of resource usage [4].*

We can conclude from the above-mentioned definitions that software miniaturization can help in moving desktop software applications onto hand-held devices. However, manual software miniaturization is tedious and error-prone process. The traceability links can help in software miniaturization to identify all the required software artefacts. As mentioned before, IT organizations do not practice requirement traceability, thus making it difficult to identify required artefact for software miniaturization process. Some traceability recovery approaches [Appendix A] aid in recovering traceability links. However, these approaches face low precision and recall problem. Thus, it becomes important to improve traceability quality to support software miniaturization.

1.1 Context: Porting Software to Hand-held Devices

Hand-held devices usage is increasing rapidly. Total sales to end users totalled 1.211 billion units in 2009 and first quarter of 2010 showed 17% increase from the same period in 2009 [Gartner]. The BRIC (Brazil, Russia, India, and China) countries are adding more and more subscriber each year. [eMarketer] forecasts that worldwide subscribership will reach 4.3 billion in 2012. As hand-held device users are increasing per year, it is becoming difficult to satisfy each user's need in term of software features. Recently Google introduced a new operating system and software development kit called Android for mobile handsets. In May 2010 Eric Schmidt, Google's chairman and CEO, revealed that company's partners are shipping about 65,000 Android handsets per day [Byrne]. If these rates were to be extended over 2011, there would be 24 million new Android-based handsets shipped by then. HTC's 1H10 forecast and Motrola's conservative full year expectations would get to 20 million units. On April 21, 2009, Apple reported that over a billion iPhone applications were downloaded in just over a year.

The growing trends towards cellular phones has created a new software industry for mobile software. Software companies provide many features in desktop applications, but it is not possible to add all these features in hand-held devices' software due to their limited storage memory, processing, and display size. Hand-held device manufacturers are trying to improve processing and increase storage memory size. However, due to increase in software features support for large software application is still lacking. Software companies are left with two options either to develop separate software for hand-held devices or manually perform software miniaturization to make their software compatible with hand held devices, though both options are time consuming and costly.

The organization Suberic.net developed the Pooka Java e-mail client relying on the Java Mail API. Pooka provides features similar to those provided by other e-mail clients, such as Thunderbird. Version 2.0 consists of 3.5 MBs of Java code. It is an open-source project, also available on SourceForge.net. Let us suppose hypothetically that an organization, for example, MobileMail, wants to offer Pooka on some hand-held devices, featuring the standard Java virtual machine on an operating system such as Microsoft Windows Mobile and Google Android.

The software miniaturization process can aid MobileMail to reduce source code size, for porting it to hand-held devices. The need for porting software to hand-held devices requires up-to-date traceability links. The traceability allows identification of the artefacts implementing a given requirement or the originating requirement of a given software artefact. However, MobileMail does not practice requirement traceability. Source code is the only source of information for MobileMail to perform tasks in hand.

It is important for MobileMail to recover traceability links between requirements and source code for performing software miniaturization. There have been many works on traceability links recovery [Appendix A]. Each approach has positive and negative aspects. For example, information-retrieval-based approach is good for automatic generation of traceability links between software artefacts. However, this approach cannot trace non-functional requirements. The goal-centric approach [5] uses softgoal interdependency graph that is useful to trace non-functional requirements. However, all these approaches have low precision and recall problems that result into low quality links. Software miniaturization depends on traceability links quality. If quality links are available then the miniaturized software will be accurate and whole process will be less error prone.

Therefore, two main issues need to be addressed, firstly, improve the quality of traceability and secondly, incorporate traceability to perform semi-automatic software miniaturization for MobileMail to port Pooka onto hand-held device. Because low quality links or wrong links can result in software maximization.

1.2 Problem: Low Precision in Automated Trace Retrieval

The goal of our research work is threefold. Firstly, it deals with identifying the factors influencing low quality traceability links. Secondly, it focuses on the controlling identified traceability quality affecting factors. Lastly, it produces a process to support software miniaturization using first and second part of this research. The main question is:

How to produce an effective artefact traceability process that can support software miniaturization?

Current traceability approaches [Appendix A] deal with recovering traceability links. However, these approaches do not handle the factors that may cause low precision and recall. We mentioned some of the factors in section 1.3.4 that may affect traceability quality. These potential traceability quality factors will be empirically verified in this research by author. We need a traceability process that can avoid or control these quality factors. In this research proposal, we introduce a traceability process. However, we will perform empirical studies to evaluate effectiveness proposed process. The main purpose of this process is to handle traceability quality factors to improve precision, recall, and provide support for software miniaturization. We have divided our main research question into following sub questions to solve the main question systematically.

1. *Do current traceability approaches support software miniaturization?*

2. *Is there any dataset available in traceability community to carry out this research?*

3. *What kind of factors can affect traceability for software miniaturization process?*

4. *How to measure the potential factors effecting traceability for software miniaturization?*

5. *How to avoid or control traceability quality affecting factors?*

6. *Can a single traceability recovery approach provide enough quality results?*

7. *How to validate the usefulness of proposed approach to support software miniaturization process?*

Sub question (*i*) will be answered via a literature review in Chapter 2. This chapter pays a special attention to explore software miniaturization and traceability issues. Question (*ii*) will be answered via Chapter 4. In Chapter 4, we will conduct an experiment to create datasets. Experiments will be conducted and literature review will be done to answer sub question (*iii*) and (*iv*). All the identified factors will be documented. The documented factors will help researcher to improve their traceability approaches and industry to improve their tools and techniques. This factor controller supports to answer question (*v*). Different available traceability approaches will be analysed to get the best of available approaches. For example, which approaches can improve precision and recall if combined together. A hybrid approach will be presented to answer sub question (*vi*). Lastly, sub question (*vii*) leads to the evaluation of the proposed model and approach both quantitatively and qualitatively. Miniaturized software will be analysed, based on performance, scalability, and portability to evaluate the usefulness of proposed approach. For example, expert will manually analyse the miniaturized software to see if proposed traceability process improves performance, scalability, or portability. If there is no difference in original and miniaturized software, then we will improve the proposed process with the help of results that expert analysed.

1.3 Research Objectives & Contribution

The above problem statement serves as a premise to establish a set of specific objectives that will constitute major milestones of this research.

1. *To create datasets that can be used for conducting experiments.* Datasets are important to carry out this research. We could not find a complete datasets for email client and instant messenger. Lack of datasets is a hurdle to perform empirical case studies. We will create two datasets for email client and instant messenger. These two datasets will be contributed to traceability community to replicate our results or perform other traceability experiments.

2. *To improve the traceability quality to support software miniaturization.* Traceability is backbone for software miniaturization. Low quality traceability affects software miniaturization process. Thus, it is important to improve traceability quality to identify all correct (user mentioned) software artefacts. We will use hybrid traceability approaches and quality factor controller to improve traceability quality.

3. *To develop a hybrid traceability approach to merge best of different approaches.* Every traceability approach has plus points. For example, goal-based traceability approach helps to trace non-functional requirements. While, IR-based approach provides functional requirement traceability. In this research, we will combine IR-based, goal-based, and value-based traceability approaches.

4. *To establish a software miniaturization approach and mechanism that cover artefact dependencies and granularity.* Different software systems can have different level of granularity. For example, Java program low granularity is Class level, because, Java is object oriented language and to create a compilable miniaturized software we need all Classes and their objects. Experts can decide the granularity level with our traceability process.

5. *Develop tools to support the proposed traceability process for making miniaturization process semi-automatic.* Our proposed process is semi-automatic. It requires tools and expert input to perform traceability. We will develop some tools that will help us to conduct empirical case studies. Developed

tools will also be provided online to other researchers. Therefore, other researchers can replicate our case studies or perform their own case studies using our tools.

6. *To demonstrate and evaluate the practicability of the artefact traceability process to support software miniaturization.* We will perform several empirical case studies to evaluate our proposed traceability process for software miniaturization. All the results will be published in conferences and journals. It will help us to improve our proposed process with the comments of reviewers.

This thesis work is being developed in the context of software miniaturization, more specifically, for the traceability improvement for miniaturization. Following are the main contributions of this research:

1.3.1 Traceability Datasets

We will create two datasets (instant messenger and email client). These datasets will contain all the requirements, manually created traceability links, and source code. The manual created traceability links will help to calculate precision and recall of the proposed approach. These datasets will be shared with the traceability community as a contribution.

1.3.2 Software Miniaturization

This research presents a semi-automatic process to support software miniaturization. This will help project managers to optimize their software for hand-held devices. Our proposed software miniaturization process name is traceability improvement for software miniaturization (TISM). TISM supports multi-objective optimization that will aid project manager to make their decisions in term of software size and customer satisfaction.

1.3.3 Hybrid Approach for Traceability

This research will provide a hybrid approach to trace functional and non-functional requirements, as well as, improve quality by combining different approaches. Functional requirement traceability helps to identify functional software artefacts. While the non-functional requirement traceability helps to identify overall software system properties, for example, performance, portability, and maintainability. Goal-based traceability approach's softgoal interdependency graph will be used to capture non-functional requirements. While IR-based approach will be combined to trace functional requirements. Lastly, value-based approach will be combined with the previous two approaches to provide economical analysis. For example whether some traceability link sets can provide benefit for the investment or not.

1.3.4 Traceability Quality Factor Controller

This research will help identify the factors affecting traceability quality. We will document these quality factors with their source and rational. For example, if the source is internal (inside the traceability process) or external (outside the traceability process) and how it affects overall traceability process. We will provide details to solve or avoid these quality factors. All the identified traceability quality-affecting factors will be documented. These quality factors documentation will aid practitioners to know potential traceability quality factors. We will develop a sub-process of TISM to handle most of the quality factors. The controller will be traceability approach independent. It will be flexible and can work with different available traceability methodologies. It will also aid researchers and practitioners to control these factors to get quality links.

Bellow are some quality factors that can cause low precision and recall. We will perform empirical case studies to verify these factors and explore new quality factors that cause low precision and recall.

Identifiers Quality Identifiers naming convention can affect traceability recovery process. If the developer has used some abbreviations or meaningless names, it will result into low quality links or no link.

Unused software modules Software teams mostly add and remove software modules. However, without proper documentation it is hard to remember which modules they should remove for future release, this result into unused software modules. When we try to recover traceability links, these unused modules will also participate in traceability relations that can result into more false positives.

Ambiguous Requirements Requirement is a dynamic concept and evolves with time. These dynamic concepts can be ambiguous, as user is not a technical person and does not know what he needs. These ambiguous requirements can take traceability process towards low quality links and false positives.

Granularity level The level of granularity should be defined before extracting traceability links. If the source artefact is class level diagram and target artefact is package level or fine grain source code granularity level then it can produce false positives and low quality links. Same for requirements, if the requirements are high level and we try to trace them into method level or source code line level then it can create problems.

1.4 Proposal Organization

This research proposal covers some discussions on the specific issues associated with traceability improvement for software miniaturization and understanding how this new research will be carried out. The proposal is organized in the following outline.

Chapter 2: Discusses the literature review of the traceability and software miniaturization. Few areas of interest are identified from which all the related issues, words and approaches are highlighted. This chapter also discuss some approaches and deployment of traceability. At the end is a discussion on literature. This leads to improvement opportunities that form a basis to develop a new process.

Chapter 3: Provides a research methodology that describes the research design, proposed process, supporting tools, and measuring methods. It is followed by some research assumptions.

Chapter 4: The proposed process is evaluated for its effectiveness and correctness. Preliminary results have been shown in this chapter and proposed process's applicability.

Chapter 5: Gives conclusion remarks about proposed TISM process.

Chapter 6: Gives directions for future work and the schedule of this research.

Appendix A: Gives details about current available traceability approaches.

Appendix B: Gives details about the importance for traceability in software engineering and application of traceability.

Chapter 2

Literature Review on Software Miniaturization and Traceability

Modifying multifaceted software systems require a detailed understanding of their functionalities. It is important to traverse the development artefacts looking for their relationship to develop this understanding. Traceability has been recognised as an important task in software system development. The requirement engineering community has done biggest part of traceability research. Traceability has gained importance and its topics have become subject to research. Traceability relations can improve the quality of the product being developed and reduce development time and cost [6].

Ramesh [7] separated traceability users into two groups, high-end and low-end. High-end users are organizations, which use traceability as well-defined system development policies. They customize their tool to provide better support for traceability. On the other hand, low-end users use traceability relations to allocate requirements to system components. However, they do not see traceability as an important task in their development process. Low-end users do not capture process-related traceability information.

Maletic et al. [8] proposed XML based traceability query language, TQL. TQL supports queries across multiple artefacts and multiple traceability link types. TQL has some primitives to allow complex queries construction and execution support. Authors executed complicated pattern matching queries using XPath on srcML and got speed of 20KLOC/sec.

Maider [9] re-focused attention on practical ways to apply traceability information models in practice to encourage wider adoption of traceability. Authors highlight the typical decisions involved in creating a basic traceability information model, suggest a simple UML-based representation for its definition and illustrate its central role in the context of a modelling tool.

Arkley et al. [10] proposed Traceable Development Contract (TDC). The TDC formalises the interaction of two development teams by defining their behaviour with respect to the state of their common development artefacts. TDC framework records traceability information to reduce incomplete, inaccurate, and out-of-date traceability links. TDC issues an upstream functional development team that impose changes on a downstream development team. The contract makes the recording of traceability beneficial to both development teams.

Cleland-Huang et al. [11] presented a model-based approach designed to help organizations gain full benefit from the traceability links they developed and to allow project stakeholders to plan, generate, and execute trace strategies in a graphical modeling environment. Sherba et al. [12] proposed an approach TraceM to traceability based on technique from open hypermedia [13] and information integration. TraceM approach manages traceability links between requirements and architecture. Open hypermedia system enables the creation and viewing of relationship in heterogeneous application. TraceM allows the creation, maintenance, and viewing of traceability relationships in tools that software professionals are accustomed to using on a daily basis.

Jirapanthong et al. [6] presented a rule-based approach to support automatic generation of traceability relations between feature-based communality and variability analysis documents. The authors defined a traceability reference model with nine different types of traceability relations for eight types of documents. This approach classifies rules into two groups: direct and indirect rules. The rules are represented in an extension of XQuery. The authors also provided the XTraQue prototype tool that supports their approach. It allows creation of new traceability rules and execution of these rules to verify their correctness. When a user is satisfied with a new rule, it can be inserted in traceability containing document.

Clealand et al. [14] proposed a heterogeneous solution TraCS (Traceability for Complex Systems) to improve return on investment (ROI) of traceability efforts.

Their hypothesis is that different types of requirements are best-traced using different methods, and that competing techniques introduce their own tradeoffs at both the individual requirement level and at the project level. This approach uses blend of manual (Simple Links) and dynamic traceability techniques (EBT, IR, SBT) in order to establish proper levels of traceability.

The authors used Softgoal Interdependency graph (SIG) to decompose traceability into sub goals to minimizes costs and maximize value. They further decomposed the goal of maximizing value into sub-goals of (i)correctness: The extent to which a set of links conform the actual relationships that exists between artefacts. (ii) Coverage: The extent to which the traceability scheme provides support for all requirements regardless of their type, geographical location, or level of abstraction. and (iii)automation: The extent to which a traceability link supports automated queries.

HRT approach also defines some strategies how to avoid conflicts between these sub-goals. It uses IR traceability techniques to maximize the dynamic link generation to avoid user-defined links. This strategy is only useful if there is high lexical correlation between artefacts. HRT uses trace for a purpose strategy to establish links that have a clearly defined purpose, and will return clearly defined value to the project. It provides support to trace same or different artefacts with different technique that performs better. The decision could be made that which links should be kept or thrown away to get ROI. The main contribution of their research is to improve ROI of requirement traceability. This technique did not present any prove that if it also helps to improve precision and recall. The main drawback of this approach is that user has to make decision which technique should be used at what time. A single wrong decision can produce wrong results.

Above-mentioned approaches, aid in creating links between functional requirements and source code. It is also worth to trace Non-functional requirements (NFRs) [15] to keep track of important aspects of software system such as, usability, flexibility, performance, interoperability, and security. NFRs play a critical role in any system. NFRs do not talk about specific module in the source code, it talks about over all system's aspects. It is difficult to trace NFR. Some researchers have tried to trace NFR.

Cleland-Huang [16] proposed goal centric traceability (GCT) that supports impact analysis of NFRs with the context of the software architecture in which they are deployed. GCT focuses on post-requirement traceability of NFRs. The authors first identify three critical areas in which NFRs require traceability support, then they evaluates existing traceability methods, and finally proposes a more holistic traceability environment named Goal Centric Traceability. The authors did not provide any empirical prove that their approach works for NFRs. Kassab et al. [17] proposed a ”language independent meta model” that explicitly models functional requirements (FR) and NFRs, their refinements, and their interdependencies. The metamodel, which is independent from any programming paradigm, is further transformed into a relational model, which facilitates NFRs traceability using tracing queries implemented through datalog expressions.

Functional and non-functional requirements' traceability help to identify software artefacts that help to perform software miniaturization. Software miniaturization is a form of software refactoring focused on reducing an application to the bare bone [4]. Some work has been done in the research area for miniaturization that mainly focuses on refactoring software.

Di Penta et al. [18] introduced the Software Renovation Framework (SRF) that uses Genetic algorithms for refactoring to remove unused objects, code clones, and miniaturized libraries. SRF works in six steps, firstly, it identifies the dependencies, secondly, it identify and remove unused functions and objects, thirdly, all duplicated and cloned source code identified, fourthly, all the circularly linked libraries are reduced or removed after identification, fifthly, large libraries are miniaturized. In the last step, objects used by many application are grouped. Authors applied SRF on geographical information system called GRASS. SRF significantly improved the software organization, reduced by about 50% the average number of objects linked by each application, and has consequently reduced the applications memory requirements.

Antoniol et al. [19] proposed an approach for moving smaller libararies via clustering and genetic algorithm, considering the initial clusters as the starting population, adopting a knowledge-based mutation function and multi-objective fitness function. They applied their approach in several medium and large-size open source software systems, such as, GRASS, KDE-QT, Samba, and MySql, allowing effectively to produce smaller, loosely coupled libraries, and to reduce memory requirement for each application.

Bodhuin et al. [20] proposed a search-based approach for dynamically re-packing downloadable applications. It exploits dynamic information to re-package java classes into jars. This approach reduces the downloading. It collects dynamic information by executing the instrumented application. Authors first generated a preliminary re-packaging by putting together resources used by common sets of scenarios, and then used genetic algorithms to improve such a re-packaging. They performed case study on medium-sized software to evaluate their approach.

Di Penta et al. [4] presented a case study focused on the refactoring of GRASS, which operates on small hand-held devices. They used, nm Unix tool, R statistical tool, and Perl script to support their approach. Their effort was aimed at software miniaturization, reducing code duplications, eliminating unused files, and restructuring system libraries and reorganizing them into shared libraries.

Antoniol et al. [21] used dynamic (keeping into account dependencies exploited during application execution in a given user profile) and static (keeping into account all possible dependencies) information to perform effective library miniaturization. Authors used hierarchical clustering techniques to identify software libraries minimizing the average executable size, followed by Genetic Algorithms to improve the identified solution. They defined multi-objective fitness function to keep low in size, at the same time, both the inter-library dependencies and the average number of objects linked (or dynamically loaded, in case of dynamically-loadable libraries) by each application. They tested their approach on medium size open source software systems.

None of the above-mentioned approaches deals with requirement traceability to perform software miniaturization. The dependency graph is the core part for making different decisions. For example, should the module be removed or not? It is quite possible, programmer has developed a module for future use and dependency graph is showing it as unused module. The requirement traceability can help in such situations. It can also help to keep track of whole software miniaturization process.

2.1 Discussion

There are many approaches available for traceability. However, none of the approach provides solution for all problems, described in Chapter 1. For example, value-based approach can be used for return-on-investment but cannot recover links, IR-based approach can be used for creating automatic traceability links between functional requirement and source code. However, IR-based approach is not effective for non-functional requirements. Goal-centric approach can provide non-functional requirements traceability but goal-centric approach requires a lot of manual work for creating softgoal interdependency graph. Table 2.1 shows the details about different approaches' support for tracing non-functional, functional, Return-on-Investment (ROI), and traceability quality affecting factor.

Software miniaturization cannot be done only by tracing functional requirements. There is a need to introduce a hybrid approach that can trace functional, non-function requirements, and provide return-on-investment analysis too, since hand-held devices are limited in non-functional features, namely performance, scalability, and portability.

Approaches	**Trace Evolution Support**	**Trace Recovery Support**	**ROI**	**FR**	**NFR**	**Quality Factor Contr.**	**Used for Soft. Miniatu**
IR-based	✓	✓	✗	✓	✗	✗	✗
Event-based	✓	✗	✗	✓	✗	✗	✗
HT-based	✓	✓	✗	✓	✗	✗	✗
Rule-based	✓	✓	✗	✓	✗	✗	✗
Scenario-based	✗	✓	✗	✓	✗	✗	✗
Value-based	✗	✗	✓	✓	✗	✗	✗
Goal-based	✗	✗	✗	✓	✓	✗	✗

Table 2.1: Traceability Approaches Summary: ✓ sign shows that current traceability approach supports above mentioned feature.✗ sign shows that current traceability approach does not support above mentioned feature.

We can see in Apendix B that traceability has different applications domains [Appendix B] where software engineer can apply traceability to get benefit for different tasks in hand. The applications of traceability show that improving traceability quality can also benefit other areas of software engineering that use traceability. All the current traceability approaches [Appendix A] focus on techniques for recovering, managing traceability links and providing ROI analysis. However, these approaches do not talk about the factors that may affect the traceability approach, process, or technique. Egyed et al. [22] mentioned about granularity and time of trace generation factors for traceability. There can be more quality factors, for example, identifiers' quality, expert knowledge, time constraints, and vague requirements. It is important to identify these quality factors and control them for improving traceability link quality.

All the current miniaturization work [4, 18, 19, 20, 21] focuses on removing unused code, code clones, and miniaturization of libraries. None of these approaches talks about users' perspective. These approaches can help to miniaturize software, but question remains, who will decide to remove or add any module. This kind of decisions can be supported with the help of requirement traceability. Manually creating traceability links and then performing miniaturization can be a costly process and error prone. Traceability is a basic step for software miniaturization. However, current approaches [Appendix A] do not provide efficient support for miniaturization.

The literature review shows that requirement traceability can be helpful in different areas of software engineering. However, there is no evidence in the literature of traceability support for software miniaturization. The main purpose to perform traceability application literature review in Appendix B was to find out if any of the researcher has used traceability for software miniaturization. We explored almost all traceability approaches in [Appendix A] to see which approach supports specifically software miniaturization or which approach can produce better results for miniaturization process. There is no proof in the literature that can show whether any work has been done in the proposed research context.

Chapter 3

Traceability Improvement for Software Miniaturization (TISM)

This chapter describes the proposed traceability improvement process to support software miniaturization. The proposed approach is established around the functional requirements, non-functional requirements, design documents, and source code. The chapter begins with a brief introduction about the proposed TISM process, followed by details of each step of this process.

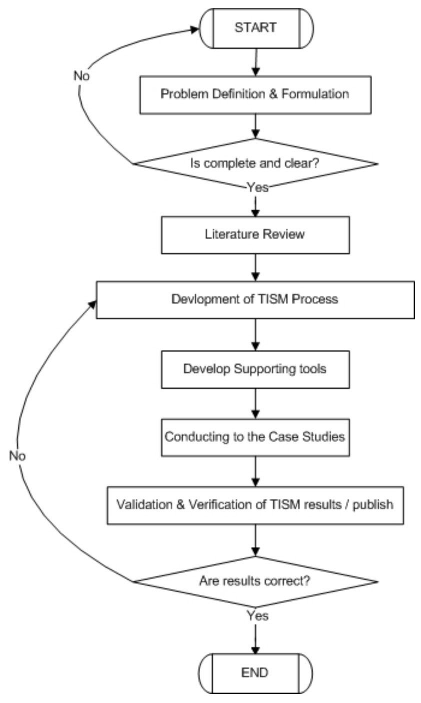

START
Problem Definition & Formulation
Is complete and clear?
No
Yes
Literature Review
Devlopment of TISM Process
Develop Supporting tools
Conducting to the Case Studies
Validation & Verification of TISM results / publish
Are results correct?
No
Yes
END

FIGURE 3.1: Research Design Flow

3.1 Research Design Flow

Figure 3.1 illustrates the operational steps of our research work. This research proposal was started by doing a high-level literature review of requirement traceability. The next step was to define and formulate the problem. During the literature review, we gathered the related information of software miniaturization and traceability to obtain some meaningful research problems. We also explored whether traceability has been previously used for software miniaturization. However, we could not find any evidence. We explored the limitation and drawbacks of the existing techniques and approaches.

The next step of our proposal was to develop a process for solving research questions. To aid our proposed process, we developed miniaturization, optimization, and traceability tools. The developed tools will be used to conduct case studies to validate and verify our proposed approach. Author will make the last decision of accepting or rejecting the proposed TISM process after peer review of TISM. If the results of case studies are not as we expected we will edit and or re-design the proposed approach. Evaluation of proposed process is an iterative process, until we achieve high precision and recall with TISM process. The findings will then be concluded, published in conferences and journals, and produced in the research thesis.

3.2 TISM Process Assumptions

The proposed TISM process makes following assumptions. If these assumptions are not true, TISM may not be able to increase precision and recall for traceability.

1. TISM process assumes that traceability recovery is for textual format. All the graphical elements' text will be manually extracted and textual part will be processed by proposed TISM process.
2. All requirements are independent but if they are dependent then backward traceability is required.
3. The datasets are complete in term of dependency. Incomplete dependency graph can still work; however, the miniaturized software will not be compilable.

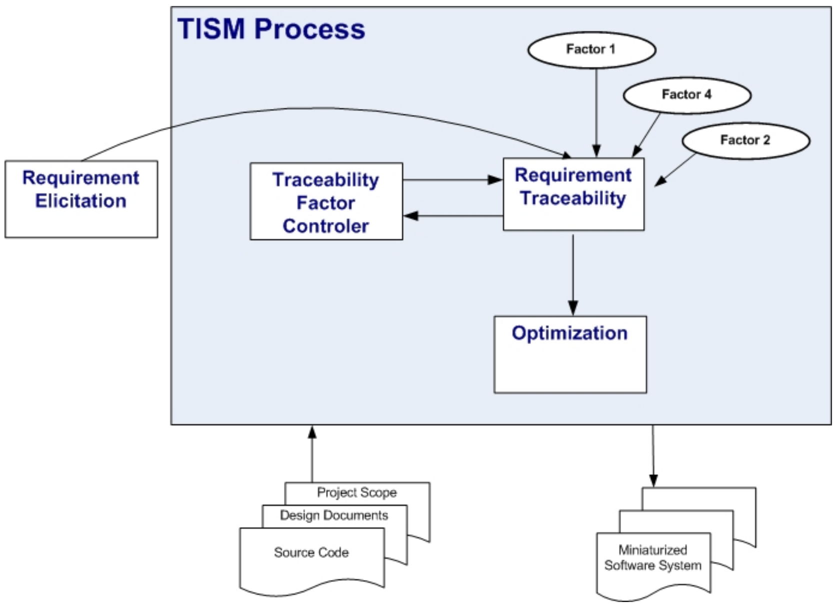


FIGURE 3.2: TISM High Level Diagram

4. Software artefacts that are used as an input for TISM are assumed to be in the correct state in term of consistency among artefacts.
5. Experts have a good knowledge of traceability, miniaturization, and optimization to perform different tasks during experiments.
6. All documents are in English.

3.3 High Level Model of TISM

In this research, we present the TISM process to improve traceability quality for software miniaturization. Figure 3.2 gives a high-level overview of TISM process. TISM is combination of sub-processes to achieve main goal that is to improve traceability quality and provide support for software miniaturization.

Requirement elicitation process is (REP) an external part of the TISM process. The REP helps to create datasets for empirical case studies. It represents an iterative set of steps in order to perform elicitation process. The REP process uses clustering techniques to gather stakeholder's understanding of the software system. The output of this process are pre-requirements that will be used in HTML as input. The main sub-process of the TISM process is Hybrid Traceability Model

(HTM). The HTM model is flexible to add and remove traceability approaches to perform traceability based on project need. The final output of the HTM is ranked list of traceability links between requirements and source code. Some factors may affect the traceability link quality. The traceability factor handler (TFH) identifies and uses precautionary measure to resolve these factors. The output of TFH is purified data that helps to increase traceability precision and recall. To remain flexible, experts can select some or all of the identified factors to resolve in this process. The HTM model can identify software artefacts that can participate in miniaturized software. However, different stakeholders want to optimize their miniaturized software with respect to size, performance, portability, and screen size. The requirement for optimization leads us to the last component of TISM process, which is miniaturization optimization (MO). The MO goal is to optimize miniaturized software based on different attributes, namely, portability, scalability, performance, and return-on-investment. Expert can make a trade-off between these optimization attributes.

Below we describe each of the sub-process of TISM in more details.

3.3.1 Requirement Elicitation

Every stakeholder can have different or similar needs for miniaturized software. However, it is not an easy task to manually collect all stakeholders' needs and group into similar and different needs. Different stakeholders can write their needs in different way. The TISM process supports semi-automatic elicitation of user needs. Users can express their needs in any textual format. All the elicited requirements will be assigned an identity. This process will group all the elicited requirements in three groups, namely, functional requirements, non-functional requirements, and outliers. The elicited requirements will be input for requirement traceability process. All the traced requirements will be used to capture required software artefacts, while new requirements (non-traced requirements) will be highlighted to project manager. Below is the technical detail of semi-automatic requirement elicitation process.

We used previously available techniques PREREQIR [23] to perform requirement elicitation process. Figure 3.3 gives the overview of requirement elicitation process. Requirement can be obtained by eliciting the stakeholders' expectations of

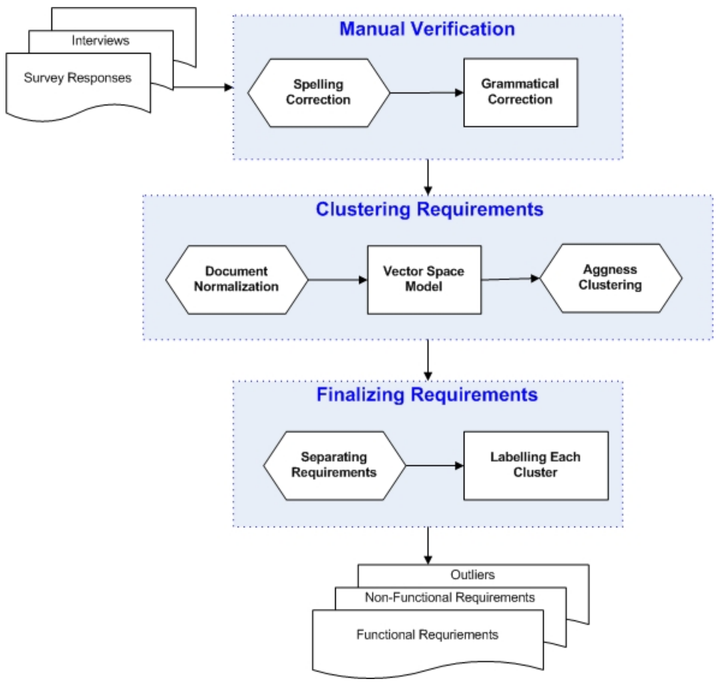


Figure 3.3: TISM - Requirement Elicitation Process

generic systems or domains. Survey and interviews can help to extract stakeholders' needs for their hand-held device software. The survey or interviews' data (raw requirements) is the input for elicitation process. Experts can then manually correct spelling and grammatical mistakes from requirements. Different available tools can be used for English spelling and grammatical corrections. The output of this step will be correct English raw requirements. These raw requirements will be used as input for clustering process. This step uses information retrieval techniques to perform similarity analysis between different raw requirements and generate similarity matrix. This similarity matrix is used by Agnes clustering to group all similar raw requirements and at the end of this process, experts can label each requirement and separate functional requirements, non-functional requirements, and outliers [23]. The output of this process will be pre-requirements and will be used in traceability process as input.

3.3.2 Artefact Traceability

Traceability process works as backbone of software miniaturization. The quality of software miniaturization depends on the quality of traceability approach. The input for traceability process is requirements and software artefacts. We have

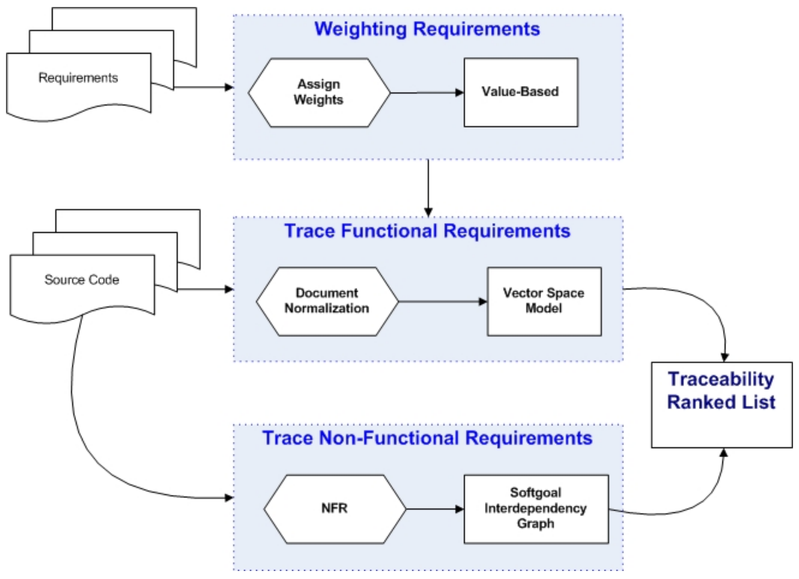


FIGURE 3.4: TISM - Traceability Process

seen in chapter 2 that there is not a single approach that can provide the facility to trace both functional and non-functional requirements. Our proposed traceability approach is hybrid. It includes IR-based, value-based, and goal-based approaches to trace requirements into source code. IR-based approach provides automatic link recovery between different software artefacts. The softgoal interdependency graph (SIG) [16] will be combined with IR-based approach to trace non-functional requirements. The non-functional requirements help to trace performance, maintainability, and portability links. Lastly, a value-based approach [24] will be combined with two other approaches. A value-based approach will help to determine which traceability links are worth to keep, and how much value it can return in miniaturization process. For example, whether keeping non-functional requirement traceability link can aid in long term and will return on investment or not. It is important for project manager to know returns on investment that they are putting in miniaturized software. It helps to manage project budget.

In traceability process expert can assign different or same weights to different requirements. This weight will aid in optimization. FacTrace online application is used to assign different weights to customers and calculating total weight of each requirement.

The output of this process is a ranked list of traceability links. These links will be used to extract all the required software artefacts from the software repository. Figure 3.4 illustrates the high-level activities and major tasks involved in our TISM traceability process. The precision and recall metrics will be used to measure the accuracy of the hybrid approach and for quality of links. Proposed hybrid approach is flexible to add or delete a traceability approach. For example, rule-based traceability approach can be integrated by experts in proposed hybrid approach to recover specific traceability links.

3.3.3 Traceability Quality Affecting Factor Handler

This process in our framework helps to answer the research question, i.e which factors can affect traceability link quality? Our hypothesis for current research proposal is that there are some factors affecting the traceability process, such as, identifiers quality, expert knowledge, ambiguous requirements, and time constraint. However, we do not know which factors can affect trace recovery more than other. We will conduct survey to know about other traceability experts opinion to determine other factors affecting traceability. We will also conduct requirement traceability case studies to verify if these factors really affect the link quality or no. These factors will be documented in the following format:

Description	Description of the factor
Rational	Source of the factors and reason why and how these factors affect link quality.
Importance	Level of the factor

Table 3.1: Traceability Quality Factor Documenting Format

Each factor will be described in detail and the source of this factor and possible rationale behind this factor. To be able to perform valid empirical research on these factors, we will use precision and recall to measure each factors' affect on traceability process. The importance of that particular factor can help to see influence of the value of that factor on link quality. IR techniques will be used to measure factors' effect.

Factor handler process will avoid any external factor and resolve the internal traceability process factors. It will handle quality factors depending on the nature of the individual factor. The process is iterative.

3.3.4 Software Miniaturization Optimization

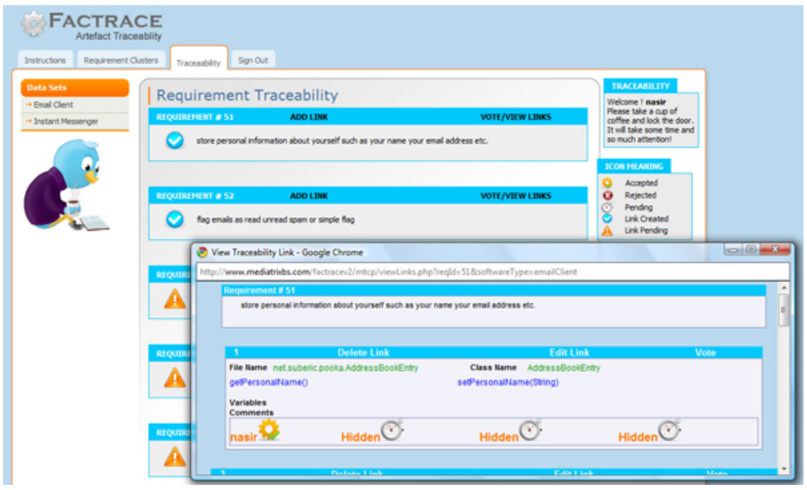
The optimization process is multi-objective. It supports following mentioned attributes for optimization. The input for this process is traced artefacts, dependency graph, and weighed requirements. Below are different attributes used by TISM to optimize miniaturized software:

Performance: The performance is critical part for miniaturized software as hand-held devices are limited in memory and processing speed. The CPU usage and memory usage will be measured. The methods that minimize CPU and memory usage will be selected and the features contributing these methods. Dynamic analysis helps to capture performance.

Scalability: The purpose of miniaturization is to maximize features with minimized source code. The least code will be select that can provide maximum features. Search-based approach will be used to draw Pareto front for expert to make the decision for scalability.

Portability: Many operating systems are available in hand-held devices. If the software is platform dependent, it will be difficult to move it to hand-held device. NFR traceability matrix will help to identify software modules that uses platform-specific constructs.

Return-on-Investment: Expert can assign weights that determine return-on-investment for optimization. Weights will be assigned to customer and their requirements. This weight helps project manager to make decision whether he wants to satisfy maximum number of customers or most important customers. Project manager can see untraced requirements weight and their customers' weight to make decision that whether it will give benefit to implement them. Search-based techniques will be used to perform optimization process. There is a trade-off between these optimization attributes. Experts can decide which optimization attributes are important to optimize software, and selected attributes will be used to provide multiple solutions to experts. The output of this process will be a Pareto front of multiple solutions. Expert can select either one or multiple solutions as per project need from Pareto front.



store personal information about yourself such as your name your email address etc.

flag emails as read unread spam or simple flag

Welcome ! **nasir** Please take a cup of coffee and lock the door. It will take some time and so much attention!

http://www.mediatrixbs.com/factracev2/mtcp/viewLinks.php?reqId=51&softwareType=emailClient

File Name net.suberic.pooka.AddressBookEntry **Class Name** AddressBookEntry

getPersonalName() setPersonalName(String)

FIGURE 3.5: Online FacTrace System

3.4 TISM Supporting Tools

This section describe the details of tools, packages, and scripts we will use for conducting experiments to support our proposed process.

3.4.1 Artefact Traceability - FacTrace

FacTrace stands for artefact traceability. FacTrace aids in recovery traceability links between source code and requirements as well as in requirement elicitation process. It has two parts: first is online and the second is a desktop application. The desktop part of FacTrace uses LSI & VSM to recover traceability links. The online FacTrace system (Figure 3.5) helps experts to remove false positive links and vote for each link. Online FacTrace helps in teamwork. Figure 3.5 shows the front-end of online FacTrace system.

3.4.2 Dependency Graph Builder

Requirement traceability can help identify all the modules that can be used in miniatruized software. However, it is quite possible those module are dependent on other modules. We need a dependency graph to extract all releated modules. For this purpose, we will use two tools AURA [25] and the NM Unix tool. The AURA only supports the Java language for dependency graph. It uses AST graph to extract all the dependencies. The NM is a unix command. It is used to examine binary files (including libraries, compiled objects modules, shared-object files, and standalone executeables) and display contents of those files and meta information. NM will be used to extract dependency graph for C/C++ source code.

3.4.3 Optimization Package

We will use genetic algorithms to support optimization part of TISM process. Genetic algorithms provide multiple solutions for single problem. We selected JMetal package to develop optimization tools to conduct experiments. jMetal stands for Metaheuristic Algorithms in Java and is an object-oriented Java-based framework aimed at the development, experimentation, and study of metaheuristics for solving multi-objective optimization problems.

3.5 Traceability Quality Measuring Methods

We will use two well-known information retrieval metrics, recall and precision, to evaluate the accuracy of our experiment results. Both measures have values in the interval [0,1].

$$Precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

Expert can define threshold value based on the project scope or retrieved documents. Precision is defined as the number of relevant documents retrieved divided by the total number of retrieved documents by an approach. Precision considers all retrieved documents above than the threshold value. This measure is called precision at n or *P@n*. If the value is 1 for precision it means that all the recovered documents are correct.

$$Recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|}$$

Recall is defined as the relevant documents retrieved divided by the total number of relevant documents. Document can be a query or result of query execution. It is ratio between the number of documents that are successfully retrieved and the number of documents that should be relevant. If the value is 1 for recall, it means all relevant documents have been retrieved.

Chapter 4

Preliminary Results

This chapter discusses the evaluation of proposed TISM process[1]. The objectives of this evaluation is to conform that the proposed process is stable. This chapter includes case study data and preliminary results using TISM process.

4.1 Experiment Goals

The goal of this case study is to miniaturize two software systems (Pooka, SIP Communicator) in terms of scalability. Miniaturized software can be optimized based on number of customers and number of valued customers.

4.2 Datasets

We use two general-public systems as case studies: Pooka [Pooka], a Java e-mail client and SIP communicator [SIP], an instant messenger. We choose these two systems because gathering requirements for such general-public systems is possible by interviewing subjects with little technical background. Our choice does not reduce the applicability of TISM to any system.

[1]This experiment is still in progress. We are preparing these results to submit in ICSE'11 for publication and will provide the submitted paper after 20th of August.

4.2.1 SIP Communicator

SIP Communicator is an audio/video Internet phone and instant messenger that supports some of the most popular instant messaging and telephony protocols such as SIP, Jabber, AIM/ICQ, MSN, Yahoo! Messenger, Bonjour, IRC, RSS and counting. SIP Communicator is distributed under the terms of the LGPL [2]. SIP Communicator is based on the OSGi [3] architecture using the Felix [4] implementation from Apache. These technologies make it very extensible and particularly developer friendly. SIP Communicator was originally created by Emil Ivov, who was at the time a student at the Louis Pasteur University in Strasbourg, France.

4.2.2 Pooka

Pooka is an email client written in Java, using the Javamail API. It supports email through the IMAP (connected and disconnected) and POP3 protocols. Outgoing mail is sent using SMTP. It supports folder search, filters, display filters, context-sensitive color encoding features. Pooka has different user interface styles, namely, Eudora and Outlook-like interfaces. Address book feature is partially supported by Pooka.

Pooka and SIP communicator datasets are currently active projects. Table 4.2.2 provide some general descriptive statistics on the two systems:

	Pooka	**SIP Communicator**
Version	2.0	1.0
Total Classes	298	1771
Total Functions	20868	31502
LOC	244870	486966
Size (in MB)	5.23	25.8

Table 4.1: Pooka & SIP Communicator General Statistics

[2]http://www.gnu.org/licenses/lgpl.html
[3]http://www.osgi.org
[4]http://felix.apache.org

4.3 Survey Population Statistics

We conducted an online survey [5] to gather requirements for email client and instant messenger. We sent 350 invitations to our colleagues and other peoples. Among the 350 recipients, only 151 responded back. Only 73 out of 151 respondents filled the whole survey. In selected 73 participants, 28 were females and 45 males. Statistics analysis of participants and their background revealed some interesting observations. Of the sample, 45.21% were student, 28.7% researchers and 26.02% industry related people. On an experience basis, 22.6% had no experience whereas 75.34% had one or more than one-year experience. Of those participated 70.08% and 80.82% did not contribute in email client and instant messenger development respectively. Concerning programming language, experience 71.23% and 72.60% had JAVA and C language experience respectively. Majority of survey participants i.e. 72.60% used Microsoft window. Out of 73 participants, 22 and 7 users do not use email client and instant messenger respectively. Survey participants spent an average time of 10.07 and 9.14 minutes to write email and instant messenger requirements respectively while 20.06 minute to complete whole survey. All the participants wrote a total 599 and 639 requirements for email client and instant messenger respectively. The population statistics shows that majority of the participants were from computer field and they have good knowledge of both software systems.

4.4 TISM Pre-Processing Step

This is pre-processing step for TISM process. We consider each requirement and source code class as a separate document. We performed the following steps for normalizing all documents. All upper-case letters were converted into lower-case letters. We removed all stop words that do not play an important in lexical similarity (such as articles, punctuation, numbers, etc.). Morphological analysis helps to convert plural into singulars and to re-conduct all the flexed verbs into the infinity form. This process is automatic and done using FacTrace. For the source code files, we performed following steps before applying above-mentioned techniques. We extracted all the identifiers from the source code. Programmers

[5]http://web.soccerlab.polymtl.ca/limesurvey/index.php?sid=28533

normally use underscore and camel-case to separate different identifier names. We used AURA [25] to extract and split identifiers.

4.5 Vector Space Model (VSM)

We used Vector Space Model [26] (VSM) to recover traceability links between source code and requirement documents. Text is represented by vectors of terms in VSM. Either a query or a document is viewed as a vector of terms. Different term weighing schemes can be used to construct these vectors. The most popular scheme is tf-idf. Term frequency (TF) is described by a t x d matrix, where t is the number of terms, and d is the number of documents in the collection. TF is often called local weight. The most frequent term will have more weight in TF but it does not mean it is an important term. It is also important to find how many times this term has occurred in whole document collection to see its importance. The inverse document frequency (IDF) of terms is calculated to measure the global weight of a term.

$$(tf - idf)_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} * \log_2(\frac{|D|}{d : —t_i \in d|})$$

$n_{i,j}$ = occurences of term (t_i) in document d_j
$\sum_k n_{k,j}$ = sum of occurrences of all terms in document d_j
$|D|$ = total number of documents in the collection
$|d : t_i \in d|$ = number of documents where the term t_i appreas

Similarity Analysis

The similarity between two vectors is not inherent in the vector space model. The angle between two vectors (A, B) is used as a measure of divergence between the vectors. Smaller the vector angle is higher the similarity between a query and a document.

$$\frac{A * B}{||A||\ ||B||}$$

In similarity analysis equation, A and B are the term frequency vectors of the documents. We have developed FacTrace tool based on VSM. It can create links between different artefacts. FacTrace has tool parts, one online system and one desktop application. The desktop FacTrace helps to recover traceability links, while online FacTrace system helps expert to verify the links and remove false positive ones. The output of this process is ranked list of links.

4.6 Requirement Elicitation

We elicited requirements for Pooka and SIP communicator to create datasets for current experiment. We used PREREQIR [23] to elicit requirements. We conducted an online survey [6] to elicit requirements for email client and instant messenger and asked participants about the features they need in their hand-held software. These features can be new or existing ones that they use in in their instant messenger and email client.

Many similar requirements were found from different participants. We use VSM and agglomerative nesting (aggnes) clustering to recover core requirements by eliminating similar requirements. VSM generates similarity matrix based on the lexical similarity between requirements. The high similar value means two requirements are same. The input for VSM is the normalized natural language requirements. Each requirement was considered as one document. VSM generated document-by-document matrix and each matrix value represents the measure of divergence between the vectors. The aggnes clustering was used to represent vectors similarity matrix. Aggness presents data in hierarchical way. It creates all singleton clusters at first and then keeps merging them to closest clusters in terms of matrix similar value. Expert manually checked the aggnes cluster tree and used 46% value as threshold. We used this value because less than this value aggnes start mixing different concepts in the same cluster. There were 235 instant messenger and 221 email client requirement clusters. Each customer was randomly weighed from 10 to 70 points. This weight will help experts to determine expert in optimization phase that which customers he wants to satisfy. Each cluster has different participants' requirements and cluster's total weight is based on participants' total

[6]http://web.soccerlab.polymtl.ca/limesurvey/index.php?sid=28533

weight. Experts used online FacTrace system to delete any duplicate clusters, separate functional and non-functional requirements, and outliers. Then each cluster was labelled by experts. Table 4.2 shows the summary of elicited requirements.

	Instant Messenger	**Email Client**
Functional Requirements	82	93
Non-Functional Requirements	20	25
Outliers	9	6

Table 4.2: Requirements Statistics

4.7 Requirement Traceability

Elicited requirements were traced into the source code. We performed document normalization step on both requirements and source code files. The VSM computes the similarity between requirements and source code documents and returns for each requirement, a ranked list of source code document. Due to dealing with Object-oriented datasets, we traced all the requirements to class level. The ranked list's high value shows there is strong relation between requirement and a class. Expert created manual links to develop oracle. This oracle can be used to evaluate traceability recovery approach. Experts spend 116 and 43 hours for creating SIP communicator and Pooka manual traceability links. Table 4.3 shows the automated traceability results. We used different threshold values, however, at 0.34 threshold value it produced better results. We made a trade-off between precision and recall. Automated process took some minutes for recovering more than 40% correct traceability links. It means we saved 67 hours of developer for creating 40% links. In addition to creating links, It is easy to discard false positive links with the help of online FacTrace system, other than creating new links. Though we can get good recall with our approach, we have to sacrifice precision.

Precision was low because the requirements in the case study were short, and many differences occur in describing the same concept for requirements written at different times. From the population statistics, we can see that majority of the respondent were non-technical who wrote the requirements. Most of them do not have software engineering experience, never participated in any kind of email client and instant messenger development. Thus, the expressing words and style can be different. Development team of datasets never communicated with the

	Pooka	**SIP Communicator**
Possible Links	11920	90321
Threshold	0.34	0.35
Automated Generated Links	1215	7412
Correct Recovered Links	144	409
Total Correct links	318	830
Precision	12%	6%
Recall	45%	40%

Table 4.3: Requirements Traceability Statistics

survey respondent. This kind of problem increase the problem space document distance from solution space artefacts. It can also cause low precision and recall in traceability recovery process.

4.8 Traceability Quality Affecting Factors

During the traceability process we found some factors which affected precision and recall. They are as follow:

Description	Identifiers quality
Rational	The names given by developer to identifiers are valuable information. They are often the starting point for the program's understanding activities, especially when high-level views, like the call graph, are not available [27]. Developers use different naming conventions for identifiers. These identifiers can help in creating automatic traceability links. However, due to personal preference, lack of time, and experience this kind of problems occurs. If developers have precise requirements, they should utilize the requirement terms while naming identifiers. In some cases, developers do not even use camel case naming convention or any specific separator for separating two different words in identifiers' name. It makes it difficult to split the identifiers' name for matching them with the requirements. For example, developers use "cmdpntr" for "command pointer" concept. It is easy for a developer to read and understand this. However, it is not easy task to separate this identifier and extract its original concept. In legacy systems, it is very difficult to update all the identifiers.
Importance	High

Table 4.4: Traceability Quality Factor (Identifiers quality)

The high importance means, it can highly affect precision and recall. We will conduct more case studies to verify these factors and identify more quality factors.

Description	Expert Knowledge
Rational	Automatic or semi-automatic traceability approaches can be as perfect as expert's knowledge. It is important expert verify the trace links and discard false positive from the recovered link. It is important for an expert to have some knowledge about the domain. For example, if it is an aerospace software system, we can not expect a metropolitan software system expert to verify aerospace system traceability links. However, both are software systems, but are different in functionality. If the expert does not have domain knowledge, it can affect the traceability results. Expert can allow some false positive links during trace verification process. We used 0 threshold to retrieve all possible links. The recall should be 100% whereas it was only 90%. It means expert had removed some good links. If the expert have good knowledge about the domain it can help in the elimination process of wrong links.
Importance	High

Table 4.5: Traceability Quality Factor (Expert Knowledge)

We will conduct survey to ask other traceability expert about their opinion for these factors and possible solution.

4.9 Dependency Graph Builder

We used AURA tool to automate extract dependency graph for a miniaturized version of each dataset. The current version of AURA works on Java at class level. It performs following steps to build dependency graph:

1. AURA builds the dependency tree of each implementation unit of projects to be miniaturized and get the size of the nodes in the tree. Here, implementation units and nodes in dependency tree can be libraries, classes or methods, depending on the programming languages (Object-Oriented or procedural) in which the projects are implemented and on granularity of miniaturizing process (at library, class or method level).

2. It uses the traceability links of each requirement generated by the previous step to identify the implementation units to satisfy it. Each requirement can trace to one or more than one class at the same time.

3. It filters out the duplicate nodes of the dependency trees of the implementation units and generates the dependency graph of the code to implement each requirement of a system.

Description	Distance between problem space and solution space
Rational	The unstructured interviews, feedbacks, and meetings can help to elicit requirements. However, a user is mostly non-technical person. He does not know about development process. He explains his requirements, as he wants. It is architect, analyst, or requirement engineer's duty to express user's need in technical way that a developer can understand. Due to shortage of time, money, and expertise, software companies do not document requirements properly, but if they do, then it is in the form of imperfect information. Imperfect information, like interpreting incomplete requirement specifications or vagueness in decisions is one of the main reasons that make software design difficult [28]. On the other hand, developers' non-interoperable tools, which evolve autonomously [29] make software structure difficult. These kinds of problems can increase the distance between problem space and solution space. It is possible final product satisfy customer's need. However, this distance between solution and problem space can create problem for creating traceability links manually or automatically. In our experiment, we saw that users expressed their requirements in different words and developers used their own knowledge and choices for writing source code, which created high false positive links.
Importance	High

Table 4.6: Traceability Quality Factor (Vague Requirements)

4. Finally, for a miniaturized version, it combines the dependency graphs of all included requirements and removes their duplicate nodes to produce the dependency graph of the project.

4.10 Optimization

Theoretically, every combination of more than one requirement of a project is a miniaturized version. Not all of them are acceptable because the numbers of satisfied customers can be too small or the code size is too large. when the project is large, it is not straightforward to make a balance between code size and the number of satisfied customers. Total number of the combination of requirements of a project is 2^n where n is the number of the requirements. Therefore, we can describe this problem as a Multi-Objective Optimization Problem (MOOP)[30, 31]. Our approach finds the Pareto front which helps the project manager to balance the source code size and the total number of satisfied customers.

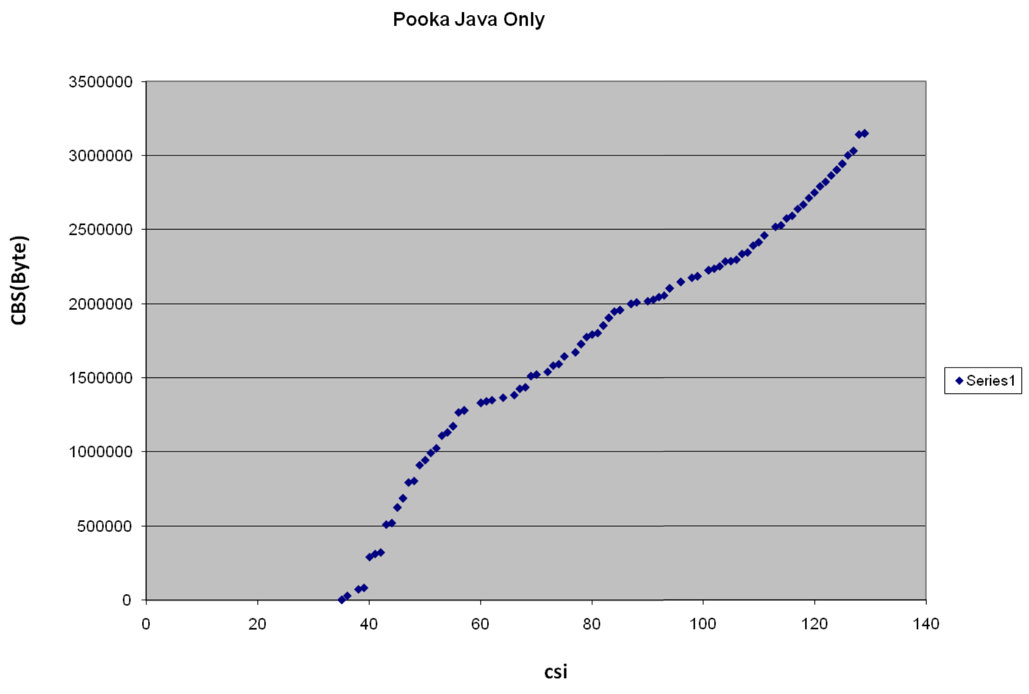


FIGURE 4.1: Pareto Front: Pooka Java Only

We assume that all requirements are independent of each other, *i.e.*, anyone of these can be added to or removed from the project without breaking the others. If there are more than one dependent requirements, we can merge them to a single requirement. To solve this problem, we apply No-dominated Sorting Generic Algorithm II (NSGAII) [32] to find one or a set of approximate Pareto fronts of all the possible solutions. We choose this algorithm because it directly generates no-dominated Pareto front with smaller computational complexity. Our approach is based on the JMetal implementation of NSGAII. The parameters of the algorithm, such as population size, crossover and mutation probabilities were the default values of the implementation.

We only included Java source code while computing the size of code base in the Pareto front. The code in the libraries or frameworks used by Pooka (Figure 4.1) and SIP Communicator (Figure 4.2) are considered as the code shared by all programs running on the device. With the help of this Pareto front graph, expert can select the optimal solution. Following are the Pareto front of Pooka and SIP Communicator:

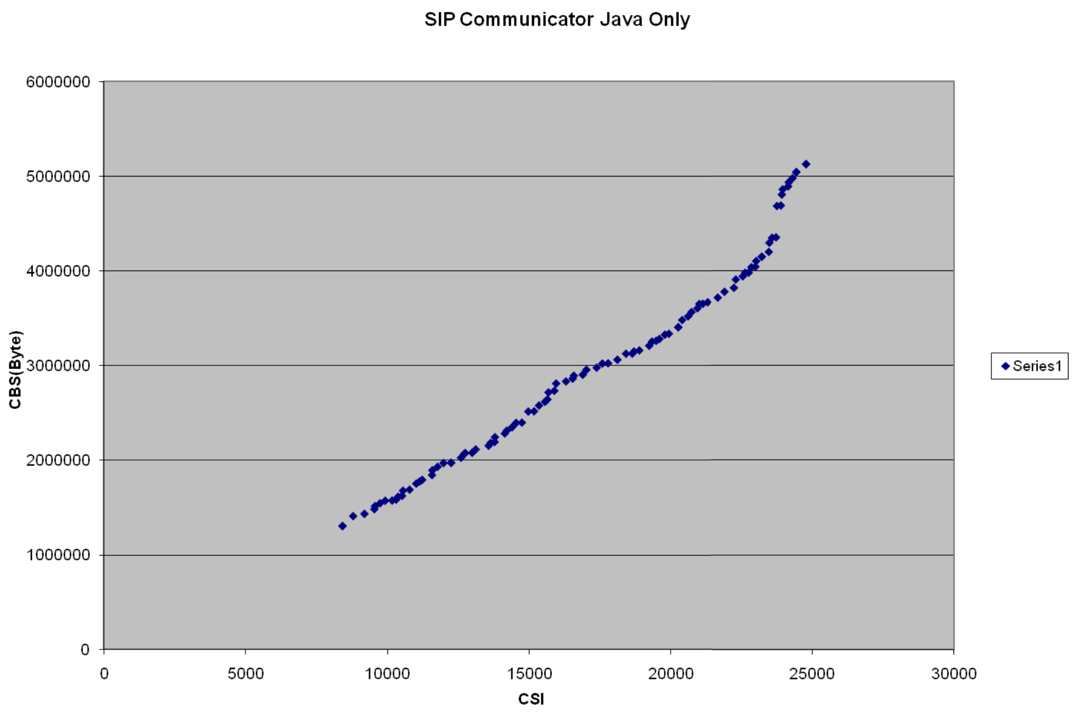


Figure 4.2: Pareto Front: SIP Communicator Java Only

4.11 Threats to Validity

There is a threat to external validity as we cannot claim that the TISM process will be effective on all software products that are to be miniaturized. To minimize this threat, we applied the approach to two different systems. The two systems examined are both open source, so we cannot claim that the approach generalizes to closed source projects. Future work will include evaluation of the TISM approach on additional systems from different domains, closed and open source.

There is a single group threat to internal validity. We did not examine the use of a control method, such as a manual approach, for miniaturizing a product. We minimized this threat by examining the amount of effort saved for human analysts at various steps of the process, such as during the traceability step.

There is a mono-operation bias threat to construct validity. The two case studies examined the miniaturization optimization when being constrained by memory. An application of TISM when constrained by a different attribute (screen size, e.g.) may yield different results. Such additional evaluations are relegated to future work. Another possible threat to construct validity is that of experimenter expectations. We attempted to mitigate this threat by involving six researchers from four different universities representing different perspectives on the research who scrutinized the results of each step of each case study.

Chapter 5

Conclusion

In this section we will look briefly at results from preliminary experiments and draw some overall conclusions. We will also look at future work in this section.

The goal of this thesis is to investigate whether traceability can support software miniaturization or not. The literature review aided in finalizing our research problem. We saw in the literature review that none of the researcher has tackled the issues that we raise in current research proposal. We also discussed some solutions for the identified research questions. For this research proposal, our hypothesis is that there is some traceability quality affecting factors. These quality factors may affect the miniaturized software system. We proposed a process (TISM) that supports software miniaturization through traceability, as well as, improving the traceability quality. TISM has different components to handle different issues. TISM's aim is to answer our main research question to aid in software miniaturization. The sub-modules of the TISM provide the solution for the sub-questions. Requirement elicitation module can aid requirement engineers to elicit requirements from different stakeholders and summarize them. The same module can also help to recover requirements for legacy systems.

The core part of TISM is traceability process. It helps in identifying the modules that can be used in miniaturized software with the help of requirements. We saw in the literature review that almost no available approaches are accurate for providing quality links. Moreover, they dont have good support for tracing non-functional requirements. Our traceability process contain hybrid approach that can help in tracing both functional and non-functional requirements. This raise the sub-research questions for traceability quality factors. For this purpose, we have included factor controller module in our process to aid improving traceability link quality. This module has two objectives, first, it document all the recovered quality factors and second provides support for avoiding or controlling any quality factor. The last module of our proposed process is optimization that can help to optimize miniaturized software system. It supports multi-objective optimization to help expert in decision-making.

To evaluate our proposed process, we conducted a case study and have presented our preliminary results chapter 4. The results show that this proposed process has potential to solve current research question. We found and documented some quality factors, which affect the traceability process during the experiment.

The proposed approach has several benefits: (i) it is a systematic approach to manage requirements and develop miniaturized software. (ii) It easily identifies requirements traceability relations and helps experts to remove ambiguity and assign identities to each requirement. (iii) Help experts to make their decision in term of software size and whether to satisfy maximum customers or valued customers. (iv) Software companies can move their big application on the hand-held devices with the help of miniaturization process. It is worth to continue our research and we aim to perform the task mentioned in Chapter 5 in the future.

Chapter 6

Future Work

The proposed TISM process in this research proposal is the basis of an on-going work to improve traceability quality and provide better support for software miniaturization process. We aim to continue this research to achieve following short and long-term goals.

6.1 Short Term Goals

Following are the short-term goals that we are planning to finish within six to eight months period time.

Systematic Traceability Literature Review: We will conduct a systematic traceability literature review to have better look inside the current state-of-the-art. It will help us to understand the life cycle and flow of the traceability, as well as, some loopholes that can be filled with our current research and future research either by us or by others.

Conducting Experiment on Two Datasets: We will conduct several experiments using our proposed TISM process to see its positive and negative aspects. Therefore, we can improve our existing process (TISM). We will use our own created datasets (email client and instant messenger) for these experiments. We will also use other datasets available in traceability community to avoid biasness.

Research paper: The results gained from experiment and case study will be submitted to ICSE'11 to have other experts' opinion about our proposed process and results. We selected ICSE to submit our result, because it is a general purpose software engineering conference and include all aspects of software engineering. Our proposed approach also has different aspects, for example, requirement elicitation, traceability, optimization, and quality attribute of traceability. We will utilize reviewers' comments and conference members comments to improve our proposed process.

6.2 Long Term Goals

Following long term goals have been planed to complete within next two to three years of this thesis duration.

Tools Creation: We have already developed some tools that helped in our case studies. However, these are not complete. We will develop more tools to make our methodology more automatic and reduce human intervention. That includes following features:

1. *Hybrid approach support in traceability link recovery process.*
2. *Support tracing non-functional requirements.*
3. *Implement automatic labelling techniques for requirements elicitation process.*
4. *Different genetic algorithms support for better-optimized solutions.*
5. *Improve performance of existing modules of FacTrace.*
6. *Traceability Quality factor controller.*

Factor Controller: In our case studies results in chapter 4, we saw that some factors exist to affect traceability results. We will design a database to store all quality factors that may also help other researchers to know these factors and provide their own solutions. We will introduce semi-automatic techniques to identify these factors and control them. If these are not controllable, it will update expert so he can avoid or manually remove them.

Hybrid Approach Traceability: There are different approaches available in the literature to recover traceability links. Each traceability approach has some benefits. We will combine IR-based, value-based, goal-centric traceability approach to trace both functional and non-functional requirements. While, value-based approach helps to perform economical analysis for investment and profit for traceability links.

Optimization: In this proposal, we used two objectives (source code size, satisfying maximum customers) for optimization. In future, we will use more objectives for optimizations including performance, scalability, satisfaction of each customer, and portability. Different kind of graphs will be used to help expert in decision-making process for optimization.

Non-Functional Requirements Traceability: It is quite possible that tracing functional requirements and missing non-functional requirements traceability leads to expensive miniaturized software in terms of memory size, processing speed, and portability. We will use existing non-functional requirements traceability approaches to trace non-functional requirements. We will also try to improve existing approaches or introduce new technique for tracing non-functional requirements and improve traceability quality.

Verification and Validation: We will conduct more case studies to evaluate our proposed TISM process. This step is incremental, if results are not good, we will improve our proposed process and tools. But if the results are good then all the results will be submitted to the conference and journal for publication. The reviewers' comments will be utilized to improve our proposed methodology.

6.3 Research Schedule

Here is an overview of research status and a time line for work accomplished and to be done in future (Table 6.1). Criteria for completed research work will be in the form of technical report, submission to a journal, or a conference. The major milestones of the proposed research are listed in a chronological order and grouped as milestones:

Milestones	Start Times	Activities
1-Literature review 2-Proposal Writing	Fall 2010 Winter 2010	+ Software miniaturization + Artefacts Traceability + Artefacts Traceability deployment + Defining TISM process + Experiments on Proposed process (TISM)
1-Comprehensive Exam 2-Proposed process Results analysis	Summer 2010	+ Experimentation in Software Engineering: an Introduction + Empirical software engineering + Evidence-based software engineering + Software testing + Software engineering + Embedded Systems + Design Patterns + Refactoring to patterns + Proposal write-up + Possible publications: + 33rd international Conference on Software Engineering (ICSE)
1-Course Work 2- TISM Process rationalization	Fall 2010	+ Scientific and Technical Communication II INF7900 + Research Methods ING6900 + Extend TISM process components details + Software engineering related Seminar
Traceability Factor Controller	Winter 2011	+ Case studies to identify traceability quality factors + Conducting survey for expert opinion about identified factors + Using tools and techniques to control quality affecting factors + Possible publications: + Book chapter in Software and Systems Traceability + 18th Working Conference on Reverse Engineering
1 - Hybrid traceability approach 2 - NFR based miniaturization	Summer 2011 Fall 2011	+ Evaluating different traceability approaches + Creating softgoal independency graph for NFR + Hybrid approach for tracing FR and NFR + Implement assessment process + Validate hybrid approach with case studies + Non-functional requirement based Software Miniaturization + Optimizing miniaturized software using genetic algorithms + Possible publications: + 34th international Conference on Software Engineering (ICSE) + 20th IEEE International Requirement Engineering Conference
Thesis	Winter 2012 Fall 2012	+ Possible publications: + IEEE Journal, Transactions on Software Engineering + ACM Journal, Transactions on Software Engineering and Methodology + Prepare for thesis write-up + Thesis write-up and defense

TABLE 6.1: Research Time Line

Following are the potential titles for our future publication that we will publish from the result of this research. OdMoMs paper is in progress and we will submit it to ICSE11.

1. *OdMoMs: On Deman Multi-objective Software Miniaturization*

2. *3D Traceability Approach for recovering Traceability Links*
3. *TraQua: Traceability Quality Factors, An Empirical Cast Study*
4. *Hybrid Traceability Approach*
5. *Non-functional Requirement Traceability to support Software Miniaturization*
6. *Imperfect Information to Perfect Traceability, A Traceability Quality Handler*
7. *Traceability Optimization for Cost-Benefit Analysis*

Appendix A

Traceability Approaches

The following sections discuss thoroughly the current traceability approaches. There are various traceability approaches available in the literature. We have mentioned only some of the most cited one.

A.1 Information-Retrieval-Based

IR approaches have been applied in domain of internet search engines for several decades and have some maturity. Many researchers have tried to use different IR approaches to establish traceability links between different software artefacts [33, 34]. IR support automatic generation of traceability links between artefacts using similarity analysis. This approach is normally based on natural languages text documents independent of structural properties. It uses an indexing process on the collection of pre-process documents (corpus). This is done in the following steps: (i) Extracting all the terms from corpus. (ii) Removing stop words and stemming (ii) Developing dictionary of all terms and number of time it occurred in that document. (iv) Calculating Term Frequency (TF) at document level. (v) Calculating Inverse Document Frequency (IDF) to reduce the weight most frequent words, at the corpus level. (vi) Developing TF/IDF weight matrix. (vii) Computing similarity between document and query to retrieve a ranked list of matched documents.

The user can use the rankings in order to understand relationships between artifacts and even requirements in order validate the links that have been generated by the IR algorithm [33, 35]. The user has to review each link to accepted or reject, this activity remains a manual task which can not be automated [36]. IR-based approach has been proposed as an after-the-fact method for traceability link recovery [37].

The most commonly used IR models applied to traceability are: Vector Space Model (VSM) [33], Probabilistic Model (PM) [38], and Latent Semantic Indexing (LSI) [34]. In each model, source type of artefacts act as query and target type of artefact act as document. For example, requirement document can be treated as query while source code as document being searched based on the query. The VSM [35] treats documents and queries as vectors in n-dimensional space. A distance or similarity function can be used to calculate similarity or distance between vectors. There is one limitation in VSM that it can not produce good results for polysemy and synonyms. The LSI [34] bases on VSM; however, it overcomes the VSM limitation. LSI uses Singular Value Decomposition (SVD) to reduce the dimensionality of LSI space. It is manual task, user has to select dimensionality reduction value, it can vary from project to project. The PM uses statistical formulas to rank documents based on probability of relevance to a query.

The effectiveness of IR technique is measured using two main metrics: recall and precision [35, 37]. For a given query: recall is the percentage of actual retrieved links and precision is the percentage of correct links as a ratio to the total number of traces retrieved. The lower precision, the manual intervention is required to review the returned results [33, 35]. IR approaches have been implemented in different tools RETRO [39] ADAMS [40] and Poirot [41].

One of the drawback for using IR is the human intervention for low precision queries. However, IR minimizes the cost to nearly zero, because human intervention is only necessary when traces are retrieved and used. However, IR traceability approaches proved it is much faster than manual approaches. It has been proved on thousands to few numbers of documents and artefacts [37]. Previous research also shown that recall and precision increase as number of documents increase [37].It can have adverse affect on time for retrieval too.

A.2 Event-Based

Clealand-huang et al. [42] proposed Event Based Traceability (EBT) approach to handle both long-term evolutionary change as well as the short-term speculative change needed to support performance related impact analysis. EBT is based upon the event-notifier design pattern [43]. EBT defines traceability relationship as publisher-subscriber relationships in which dependent artefacts must subscribe to the requirements on which they depend [44]. EBT approach involves three main components, i.e. (i) Requirement manager, (ii) Event server, and (iii) Subscriber manager. When a change occurs in requirement (merge, replacement, refinement, or abandonment) an event notification message is published. The event server receives event notification and forward to all dependent artefacts. A subscriber manager is responsible for listening all incoming event notification for a set of similarly typed artefacts on the behalf of the subscribers that it manages for event notification forwarded by the event server. In some cases, the event manager may automatically update the managed object, whilst in other cases the event message must be stored in an event log to await manual intervention [42]. EBT also provides mechanism needed to support the performance related impact analysis. In EBT, The main advantage of EBT is that when changes are introduced into the system, they can be adequately propagated throughout the system of artefacts [45].

EBT supports two type of changes functional and contextual change. Functional changes occur when a new requirement is added in the system. Functional changes involve functionality and performance that is why they are hard to predict. If a quantitative change occur in existing requirement is called contextual change. The contextual changes update EBT system and execute automated re-execution of relevant performance models with the new values from the change. It enables project managers to analyze that should they implement new changes or no, and helps developers to understand change effects.

The EBT assume the user has established the traceability links among artefacts. Once links are established, changes can then be monitored [45]. EBT focused only to manage up-to-date traceability links based on changes that may occur. Clealand-huang et al. [42] made prototype based on EBT to manage and maintain traceability between requirements and UML artifacts as well as test cases.

A.3 Hypertext-Based

Metalic et al [46] proposed hypertext-based (HBT) approach for traceability between different artefacts. HBT allows traceability link recovery as well as versioning of traceability links. Sherba et al. [12, 46] also proposed hypertext-based traceability link generation using open hypermedia and information integration. Metalic et al [46] approach centers LSI and it is partially automated. However, user input is necessary for different task (selecting source code, documents, selecting dimensionality subspace of LSI, and determine threshold value for traceability) during HBT process. HBT treats each file as document and if files are too large it break down into parts roughly the size of average document in the corpus.

The documents are broadly divided into two categories: conformance and non-conformance. Conformance category Is further divided into two types: causal and non- causal conformance. The causal relationship carry implied logical ordering of document involved while a non-causal conformance relationship exist when documents or part of documents must agree each other. The causality can not be clearly identified in non-causal relations. Non-conformance links are intended to support navigational and organizational relationships that have little or no relevance to determine agreement between documents.

In HBT model, links have two properties: arity (link specifies the number of its endpoints) and directionality (navigational and logical links). Navigational directionality refers to the directions in which a link may be traversed. Logical directionality is a semantic quality that is independent of how a link can be traversed. HBT heavily depends on separating navigational and logical directionality. In HBT, models and created links have been represented in XML format. This approach categorize model into two types: anchor model and target model. When the traceability links have been created between models. A meta-differencing mechanism can be used to monitor changes in models that may occur during software evolution. HBT supports evolution by fine-grained versioning technique.

A.4 Rule-Based

Spanoudakis et al [47] presented a rule-based approach to extract traceability links between requirement statement documents (RSD), use case documents (UCD), and analysis object models (AOM). They used mainly two rules, i.e. requirement-to-object model (RTOM) and inter-requirement (IREQ). These rules were deployed on previously mentioned artefacts.

RTOM rules are used to trace syntactically related terms in the textual parts of RSD and UCD to an AOM. When RTOM rule matches found, a rule specified type traceability link is created. The syntactic traces are defined as patterns of terms that have specific grammatical roles in piece of text. These grammatical roles are determined by probabilistic grammatical tagging technique [48]. IREQ rules are used to trace between RSD and UCD. These traces are called derived traceability as they are generated between parts of the RSD and UCD. It will only be created, if these parts are traced to elements of AOM by RTOM traceability links.

This technique assumes that all of document types are in XML-based format. The traceability rules are also represented in XML-based mark-up language. Rule-based traceability consists of four stages. (i) grammatical tagging of UCD and RSD using CLAW [48] (ii) converting UCD, RSD, and AOM into XML representations (iii) generating traceability links between UCD, RSD, and AOM, and (iv) the generation of traceability relations between different parts of the requirement statement and use case documents.

Nentwich et all [49] developed rule-based tool (xlinkit) for consistency management for xml-based software artefacts.Xlinkit assumes all artefacts are in document-objece-model (DOM) tree format. It use first-order logic to describe consistency rules. This approach has a fine-grained level of granularity since links can be created at a method level in the source code (in the xlinkit too [49]). RB approach also does not directly invoke into executable element of the software when applying the rules, since the rules are only invoked into RSD, UCD, and AOM. RB technique does not support partial generation for artefacts evolution. If change has been made to a particular artefact, then RB technique will re-generate traces as it generates links for the first time. RB technique can support versioning since the links regeneration is independent from the previously established links.

A.5 Scenario-Based

Egyed et al. in [50] proposed scenario-based (SB) traceability approach. SB has three pre-requisite in order to create traceability links between artefacts. First, there must be an executable or simulate able software system, it can be partial implementation or incremental prototype. Second, there must be a software model for the system. Finally, there must be test cases or scenarios that can be executed.

Various types of test scenarios can be used to observe and record the activities of the system. Since those observations correspond directly to scenarios this implies that trace information is established between those scenarios and the system. It is not important for SB whether scenarios are tested manually or automatically, as long as some automatically trace observation tools are used to observed footprints. Footprint mean the classes that were executed while testing a scenario [51].

Trace analyzer [50] can be used to create the links between artefacts. Third party tools such as Rational PureCoverage [IBM] can be used to observe the footprints. SB approach can validate the following four types of traces: (a) traces between scenarios and system, (b) traces between model elements and system, (c) traces between scenarios and model elements, and (d) traces between model elements.

SB approach consists of four major activities called Hypothesizing,Atomizing, Generalizing, and Refining.

Hypothesizing is used to reason about potential trace information. Hypothesizing is the manual activity of SB approach. SB requires limited amount of hypothesized trace information, otherwise, the cost of using it can be too high. SB assumes that most are latest partially correct. However, it can target the wrong traces and create new correct traces. These traces with automatic new generated traces can be used for further reasoning.

SB uses footprint graph to reason about traces and how traces relate to model elements. Footprint graph also provides foundation for the generalizing and refining activities of SB. Footprint graph can built by adding new scenario in the graph, it involves two steps. First, identify the largest node that is either equal to or part of the new scenario. If two nodes are equal they are merged together and if one node has child then a parent-child relationship is created. In case no node is found or in case a found node only partially covers the footprint of the new scenario. Second, identify smallest node that overlap with the new scenario.

In generalizing step, pure hierarchical decomposition of the footprint graph is done. This hierarchy is then traversing the footprint graph starting at its leaves to propagate trace information to their parents. Refining, the reverse activity, and then traverse the footprint graph starting at its roots to propagate trace information to its leaves [51]. SB can aid to reduce ambiguity in manual links by introducing dynamic verification in large systems. SBT is a viable solution for requirements change management activities as well as impact analysis of changing requirements [IBM].

A.6 Value-Based

Zemont [24] introduces a framework for assessing the value that requirement traceability can provide to an organization in order to support decisions related to the implementation of requirement traceability.

The value-based (VB) approach provides a technical model and an economic model for requirements tracing, depending on criteria like number of requirements, value of requirements, risk of requirements, number of artefacts, number of traces, precision of traces, size of artefacts, cost/effort of trace identification and maintenance, and value of traces [52]. VB consist of five processes. (i) Requirement definition: project manager or requirement engineer analyse each requirement and assign a unique identifier to create requirement list with id. (ii) Requirement prioritization: all stakeholders asses the requirement and divide them on three priority levels based on value, risk and effort of each requirement. (iii) Requirement packaging: identification of requirement cluster to define and refine architecture from a given set of requirements. (iv) Requirement linking: project team establish traceability links between requirements and different software artefacts. Important requirements are traced in detail while less important requirements are traced at less detail to create overall traceability plan. (v) Evaluation: project manager can use traces to estimate the impact of change requirements.

In VB cost and benefits of requirements tracing mainly depends on project context, tailoring parameters, and cost and benefits. VB combines a manual and semi-automated way in obtaining the traceability links and in performing a change in the software artefacts. VB does not support versioning feature since the links are built in manual way. The scope of VB technique includes high-level and low-level artefacts. This technique has a fine-grained level of granularity since links can be created at a method level in the source code. VB approach provides traceability analysis as well as change impact analysis to support software evolution.

Egyed et al. [22] descirbed how VB technique can be used to determine (a) the level of detail of traces among artifacts (package, class, method levels); (b) the value of the artifacts that are traced (high-value artifacts justify a higher level of tracing effort); and (c) the points in time of trace generation (early vs. late).

Appendix B

Applications of Traceability

The following sections discuss the deployment of traceability in software engineering. The main purpose to do traceability usage literature study is to see if any of researcher has used traceability for software miniaturization or not. This section also shows the importance of traceability in the different domains of software engineering.

B.1 Impact Analysis and Concept Location

Impact analysis is a process of identifying the potential consequences of a change, and estimating what must be modified to implement that change [53]. One of the applications of requirements traceability is the change impact analysis [54]. Many researchers have used traceability to analyze change impact. David et al. [54] focused on requirement and requirement relations from traceability perspective to provide formal definition of the requirement relations in SysML [OMG] for change impact analysis. Cleland-Huang et al. [44] used an event-based approach to establish dynamic traceability links between requirements and performance models of the software system. Quantitative values in performance related requirements are adjusted to reflect proposed changes and impacted models are re-executed to measure the impact of the change. The resulting outputs are then automatically compared to relevant performance requirements and a system-wide report showing the impact of the proposed change upon performance is generated [45]. Knethen [55] proposed the TraceChange approach for impact analysis that focuses on functional system requirements changes of embedded control system. TraceChange consists of three parts: (1) a fine-grained conceptual trace model, (2) process descriptions of how to establish traces and how to analyze the impact of changes, and (3) supporting tools [55]. Ibrahim [56] presented a software traceability approach to support change impact analysis of object-oriented software. It provide the ability to integrate the high-level software models with low-level software models that involve requirements, test cases, design, and code. It allows a direct link between same and different level components. It supports top-down and bottom-up traceability in response to tracing for the potential effects [56].

B.2 Maintenance

Large software systems have large number of elements and their integration is complex. The understanding of large software systems is required for maintenance task. The design rationale can support such understanding but it is often undocumented or unstructured. The absence of design rationale makes the maintenance task more difficult. Traceability relations can be used to recover relations between different artefacts to understand their design rationale.

Antoniol et al. [35] proposed a method based on information retrieval techniques to trace the maintenance request text to software components that can be affected directly with this maintenance request. Di Lucca et al. [57] applied different information retrieval techniques and machine learning approaches to trace incoming maintenance request to the specialized maintenance team. They used training set of correctly classified maintenance request. This trained data helped new incoming request to go directly to specialized maintenance team. They used some distance metrics to measure the distance between team specialization and new incoming maintenance request.

Tang et al. [58] proposed rationale-based architecture model that incorporates design rationale, design objects, and their relationship. This model provide reasoning that why design objects exists and their dependencies. Authors applied traceability on this model to help several maintenance activities such as change impact analysis and root-cause analysis. It helps software architects to better understand and reason about software architecture. Their model uses UML representation. They also developed a prototype for tracing of their model.

B.3 Program Comprehension

Requirement traceability creates and utilizes relations between stakeholders requirements and artefacts produced during the software development lifecycle. Traceability relations between artefacts and manual pages assist in top-down and bottom-up program comprehension [59]. Tracing feature evolution across releases and identifying feature interaction reduce the program comprehension effort [60]. Egyed et al. [51] presented tool-supported approach that requires the designer to specify some trace dependencies but eases trace acquisition by generating remaining traceability links automatically. They illustrated their approach using a video-on-demand system to show that generated traces can be used in software engineering scenarios to improve software understanding. Ratanotayanon et al. [61] presented an approach for providing software comprehension support in Agile software development using traceability relations. Authors also presented their prototype tool, Zelda. Zelda is an Eclipse [Eclipse] plug-in that helps developers to create links from user stories and tasks to source code, test cases, and various text-based files. Zelda is designed to work with Agile practices that captures and maintains links between high-level information and source code. This approach leverage the sequence in which artefacts are created in Agile development to provide a lightweight, code centric means to capture links between user stories, test cases, and source code. De Lucia et al. [62] developed an Eclipse [Eclipse] plugin COCONUT (COde COmprehension Nurturant Using Traceability) capable of indicating the traceability links between high-level artefacts and the code under development to improve Comprehensibility of Source Code. COCONUT uses information retrieval techniques to create traceability relations. It calculates similarity between source code components and high-level artefacts. Similarity suggests actions aiming at improving the correct usage of identifiers and comments in source code to improve traceability and the comprehensibility level.

B.4 Software Reuseability

Software artefacts at different level of abstraction can be identified and reused with the help of different traceability relations. As software evolves, the documentation becomes outdated. It results in difficulty to understand the current system and reuse existing modules. Therefore, a successful reuse strategy must propose a solution to the traceability gap between software components and stakeholders requirements [63]. The ReDSeeDS project use similarity of requirement specifications for software reuse. The software repositories, such as SVN, are used to save time for software reusability. In order to reuse the existing software part in repository, ReDSeeDS creates traceability links between new requirements and old software, saved in repository. It saves a lot of development time and money [64]. It assumes that solutions for similar old requirements are reusable in new software case [64]. Hong [65] presented the Architecture-centric Software Process for Pattern Based Software Reuse approach. It uses architecture-centric software process that combines architecture-centric modeling approach with software components development process. It builds traceable component model. The architecture patterns are used as the development expertise and are the key factors in software reuse. It explicitly creates traceability links between components [65]. The traceability mode can be used to identify reusable components in the existing software system.

B.5 Software Verification and Validation

Traceability links can provide a start for performing verification and validation, to ensure that system implemented whole user's requirements and validate certain properties and specification. Requirements traceability analysis is widely considered the most effective software V&V methods [66]. Koo et al. [67] developed NuSISRT (Nuclear Software Inspection Support and Requirements Traceability) tool and integrated requirement traceability to help verification and validation (V&V) of software systems. NuSISRT traceability view supports traceability analysis between two documents. This view helps users to assess coverage for verification and validation. Hayes et al. [68] presented REquirements TRacing On-target (RETRO), requirement traceability tool. RETRO also helps in V&V and is used in both research field and industry [68]. Deng et al. [69] proposed a traceability technique named RBC (Retrieval By Construction). Authors preliminary investigations suggest that RBC can be a useful traceability technique for validating and verifying UML formalisations. This approach creates traceability links between UML model and target model representing UML semantics by using generative procedures. A generative procedure determines which elements should be created in a target model to formalise an element in UML model.

B.6 Testing

Software testing is the central means for ensuring that system performs as expected. Traceability links can be used to check the existence of appropriate test cases to validate different type of requirements and use such cases for further testing. Traceability is required to support activities such as result evaluation, regression testing, and coverage analysis [70].

Bouquest et al. [70] proposed an approach for automated test case generation for smart card software validation. Authors developed the LEIRIOS Test Generator too that support their approach. This approach expects formal models (written with B specification language) to be annotated with requirements identifiers. When the test cases are generated from the models, the identifiers are used to create the traceability matrix relating requirement identifiers to test cases identifiers.

Naslavsky et al. [71] proposed an approach that influences model transformation traceability technique to create links between artefacts. This approach adopts high-level models as the basis for test generation. It also includes the creation and maintenance of traceability relations between test-related artefacts. Traceability is required to support activities such as coverage analysis, evaluation, and regression testing. Model-driven development and model transformation solutions address the traceability problem by creating relationships among transformed artefacts throughout the transformation process. Their fine granularity enables the support for result evaluation, coverage analysis and regression testing.

Naslavsky et al. [72] proposed a model-based testing approach that uses model transformation techniques to create a traceable infrastructure that comprises model-based testing artefacts and fine grained relationship among these models. Their approach supports model-based regression testing by leveraging fine-grained relationship established during test generation process.

Bibliography

[1] E. Iee. Ieee std 610.12-1990(r2002). *IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[2] A. Rawashdeh and B. Matalkah. A new software quality model for evaluating cots components. *Journal of Computer Science*, 2(4):373–381, 2006.

[3] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001. ISSN 0098-5589.

[4] M. Di Penta, M. Neteler, G. Antoniol, and E. Merlo. Knowledge-based library re-factoring for an open source project. In *WCRE '02: Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02)*, page 319, Washington, DC, USA, 2002. IEEE Computer Society.

[Gartner] Gartner. Gartner says worldwide mobile phone sales grew 17 per cent in first quarter 2010. URL http://www.gartner.com/it/page.jsp?id=1372013. [Online; accessed 08-June-2010].

[eMarketer] eMarketer. How big will mobile get? URL http://www.emarketer.com/Article.aspx?R=1006855. [Online; accessed 08-June-2010].

[Byrne] Gavin Byrne. Google's momentum is growing in the mobile handset market. URL http://www.informatm.com/itmgcontent/icoms/s/sectors/handsets-devices/20017778004.html. [Online; accessed 08-June-2010].

[5] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina. Goal-centric traceability for managing non-functional requirements. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 362–371, New York, NY, USA, 2005. ACM. ISBN 1-59593-963-2.

[6] W. Jirapanthong and A. Zisman. Xtraque: traceability for product line systems. *Software and Systems Modeling*, 8(1):117–144, February 2009. ISSN 1619-1366.

[7] B. Ramesh. Factors influencing requirements traceability practice. *Commun. ACM*, 41(12):37–44, 1998. ISSN 0001-0782.

[8] J. I. Maletic and M. L. Collard. Tql: A query language to support traceability. In *TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability*

in Emerging Forms of Software Engineering, pages 16–20, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3741-2.

[9] P. Mader, O. Gotel, and I. Philippow. Getting back to basics: Promoting the use of a traceability information model in practice. In *TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 21–25, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3741-2.

[10] P. Arkley and S. Riddle. Overcoming the traceability benefit problem. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 385–389, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2425-7.

[11] J. Cleland-Huang, J. H. Hayes, and J. M. Domel. Model-based traceability. In *TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 6–10, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3741-2.

[12] S. A. Sherba, Kenneth M. A., and M. Faisal. A framework for mapping traceability relationships. In *2nd International Workshop on Traceability in Emerging Forms of Software Engineering at 18th IEEE International Conference on Automated Software Engineering*, pages 32–39, Montreal, Quebec, Canada, 2003.

[13] K. Osterbye and U. K. Wiil. The flag taxonomy of open hypermedia systems. In *HYPERTEXT '96: Proceedings of the the seventh ACM conference on Hypertext*, pages 129–139, New York, NY, USA, 1996. ACM. ISBN 0-89791-778-2.

[14] J. Cleland-Huang, G. Zemont, and W. Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 230–239, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2174-6.

[15] M. Glinz. On non-functional requirements. *Requirements Engineering, IEEE International Conference on*, 0:21–26, 2007. ISSN 1090-705X.

[16] J. Cleland-Huang. Toward improved traceability of non-functional requirements. In *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 14–19, New York, NY, USA, 2005. ACM. ISBN 1-59593-243-7.

[17] M. Kassab, O. Ormandjieva, and M. Daneva. A metamodel for tracing non-functional requirements. In *CSIE '09: Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering*, pages 687–694, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3507-4.

[18] M. Di Penta, M. Neteler, G. Antoniol, and E. Merlo. A language-independent software renovation framework. *J. Syst. Softw.*, 77(3):225–240, 2005. ISSN 0164-1212.

[19] G. Antoniol, M. Di Penta, and M. Neteler. Moving to smaller libraries via clustering and genetic algorithms. In *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pages 307 – 316, 2003.

[20] T. Bodhuin, M. Di Penta, and L. Troiano. A search-based approach for dynamically re-packaging downloadable applications. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 27–41, New York, NY, USA, 2007. ACM.

[21] G. Antoniol and M. Di Penta. Library miniaturization using static and dynamic information. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*, page 235, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1905-9.

[22] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher. A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 2–7, New York, NY, USA, 2005. ACM. ISBN 1-59593-243-7.

[23] J. H. Hayes, G. Antoniol, and Y. G. Guéhéneuc. Prereqir: Recovering pre-requirements via cluster analysis. *Reverse Engineering, Working Conference on*, 0:165–174, 2008. ISSN 1095-1350.

[24] G. Zemont. *Towards Value-Based Requirements Traceability.* PhD thesis, DePaul, USA, 2005. Adviser-Chang, Carl.

[25] W. Wu, Y. G. Guéhéneuc, G. Antoniol, and M. Kim. Aura: a hybrid approach to identify framework evolution. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 325–334, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-719-6.

[Pooka] Pooka. Pooka email client. URL http://www.suberic.net/pooka. [Online; accessed 13-May-2010].

[SIP] SIP. Sip communicator. URL http://www.sip-communicator.org. [Online; accessed 23-May-2010].

[26] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, New York, 1999.

[27] B. Caprile and P. Tonella. Restructuring program identifier names. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, page 97, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0753-0.

[28] J. Noppen, P. B. van den, and M. Aksit. Software development with imperfect information. *Soft computing*, 12(1):3–28, June 2008.

[29] A. Zisman, G. Spanoudakis, E. Perez-Minana, and P. Krause. Towards a traceability approach for product families requirements. In *Proceedings of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications*, Orlando, USA, May 2002.

[30] A Osyczka. Multicriteria optimization for engineering design. *Design Optimization*, pages 193–227, 1985.

[31] Y. Sawaragi, N. Hirotaka, and T. Tetsuzo. *Theory of multiobjective optimization.* Academic Press, Orlando :, 1985. ISBN 0126203709.

[32] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: Nsga-ii. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 849–858, London, UK, 2000. Springer-Verlag. ISBN 3-540-41056-2.

[33] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002. ISSN 0098-5589.

[34] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.

[35] G. Antoniol, G. Canfora, G. Casazza, and A. de Lucia. Identifying the starting impact set of a maintenance request: A case study. In *CSMR '00: Proceedings of the Conference on Software Maintenance and Reengineering*, page 227, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0546-5.

[36] J. H. Hayes and A. Dekhtyar. A framework for comparing requirements tracing experiments. *International Journal of Software Engineering and Knowledge Engineering*, 15(5):751–782, 2005.

[37] J. H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *RE '03: Proceedings of the 11th IEEE International Conference on Requirements Engineering*, page 138, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1980-6.

[38] G. Antoniol, G. Canfora, A. de Lucia, G. Casazza, and E. Merlo. Tracing object-oriented code into functional requirements. In *IWPC '00: Proceedings of the 8th International Workshop on Program Comprehension*, page 79, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0656-9.

[39] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 249–259, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2174-6.

[40] A. Lucia, De, F. Fasano, R. Oliveto, and G. Tortora. Adams re-trace: A traceability recovery tool. In *CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, pages 32–41, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2304-8.

[41] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 356–357, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2555-5.

[42] J. Cleland-Huang, C. K. Chang, G. Sethi, K. Javvaji, H. Hu, and J. Xia. Automating speculative queries through event-based requirements traceability. In *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 289–298, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1465-0.

[43] S. Gupta, J. Hartkopf, and S. Ramaswamy. Event notifier: A pattern of event notification, 1998.

[44] J. Cleland-Huang. *Robust requirements traceability for handling evolutionary and speculative change*. PhD thesis, Chicago, IL, USA, 2002. Adviser-Chang, C.

[45] J. Cleland-Huang, C. K. Chang, and J. C. Wise. Automating performance-related impact analysis through event based traceability. *Requir. Eng.*, 8(3): 171–182, 2003.

[46] S. A. Sherba. *Towards automating traceability: an incremental and scalable approach*. PhD thesis, Boulder, CO, USA, 2005.

[47] G. Spanoudakis, A. Zisman, E. Prez-Miana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105 – 127, 2004. ISSN 0164-1212.

[48] G. Leech, R. Garside, and M. Bryant. The tagging of the british national corpus. *15th International Conference on Computational Linguistics (COLING 94)*, page 622628, 1994.

[49] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Trans. Internet Technol.*, 2(2):151–185, 2002. ISSN 1533-5399.

[50] A. Egyed and P. Grünbacher. Automating requirements traceability: Beyond the record & replay paradigm. In *ASE'02: Proceedings of the 17th IEEE international conference on Automated software engineering*, page 163, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1736-6.

[51] A. Egyed and P. Grünbacher. Supporting software understanding with automated requirements traceability. *International Journal of Software Engineering and Knowledge Engineering*, 15(5):783–810, 2005.

[IBM] IBM. Rational software. URL http://www.rational.com. [Online; accessed 23-May-2010].

[52] M. Heindl and S. Biffl. A case study on value-based requirements tracing. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 60–69, New York, NY, USA, 2005. ACM. ISBN 1-59593-014-0.

[53] L. C. Briand, Y. Labiche, L. O'Sullivan, and M. M. Sówka. Automated impact analysis of uml models. *Journal System Software*, 79(3):339–352, 2006. ISSN 0164-1212.

[54] D. ten Hove, A. Göknil, I. Kurtev, K. G. van den Berg, and K. de Goede. Change impact analysis for sysml requirements models based on semantics of trace relations. In J. Oldevik, G. K. Olsen, T. Neple, and D. Kolovos, editors, *Proceedings of the ECMDA Traceability Workshop (ECMDA-TW) 2009, Enschede, the Netherlands*, pages 17–28. Centre for Telematics and Information Technology, University of Twente, June 2009.

[OMG] OMG. Sysml specification. URL OMGptc/06-05-04http://www.sysml.org/specs.htm. [Online; accessed 23-May-2010].

[55] A. V. Knethen. Change-oriented requirements traceability: Support for evolution of embedded systems. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 482, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1819-2.

[56] S. Ibrahim and N. B. Idris. A requirements traceability to support change impact analysis. *Asean Journal of Information Technology, Pakistan*, pages 345–355, 2005.

[57] G. di Lucca. An approach to classify software maintenance requests. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 93, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1819-2.

[58] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6):918 – 934, 2007. ISSN 0164-1212.

[59] A. De Lucia, R. Oliveto, and G. Tortora. Adams re-trace: traceability link recovery via latent semantic indexing. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 839–842, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-079-1.

[60] G. Antoniol, E. Merlo, Y. G. Guéhéneuc, and H. Sahraoui. On feature traceability in object oriented programs. In *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 73–78, New York, NY, USA, 2005. ACM. ISBN 1-59593-243-7.

[61] S. Ratanotayanon, S.E. Sim, and R. Gallardo-Valencia. Supporting program comprehension in agile with links to user stories. In *Agile Conference, 2009. AGILE '09.*, pages 26 –32, aug. 2009.

[Eclipse] Eclipse. Eclipse jdk. URL http://www.eclipse.org. [Online; accessed 23-May-2010].

[62] A. De Lucia, R. Oliveto, F. Zurolo, and M. Di Penta. Improving comprehensibility of source code via traceability information: a controlled experiment. In *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension*, pages 317–326, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2601-2.

[63] S. B. D. Dakhli. The solution space organisation: Linking information systems architecture and reuse. In *Information Systems Developmen*, pages 101–109, Springer US, September 23 2009. Springer-Verlag. ISBN 978-0-387-84810-5.

[64] D. Bildhauer, T. Horn, and J. Ebert. Similarity-driven software reuse. In *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, pages 31–36, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3714-6.

[65] W. Hong. Architecture-centric software process for pattern based software reuse. In *WCSE '09: Proceedings of the 2009 WRI World Congress on Software Engineering*, pages 95–99, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3570-8.

[66] S. R. Koo, P. H. Seong, J. Yoo, S. Deok Cha, and Y. J. Yoo. An effective technique for the software requirements analysis of npp safety-critical systems, based on software inspection, requirements traceability, and formal specification. *Reliability Engineering & System Safety*, 89:248 – 260, 2005. ISSN 0951-8320.

[67] S. R. Koo, P. H. Seong, J. Yoo, S. Deok Cha, C. Youn, and H. Han. Nusee: An integrated environment of software specification and v&v for plc based safety-critical systems. *Nuclear Engineering and Technology*, 38:259276, 2006.

[68] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, E. A. Holbrook, S. Vadlamudi, and A. April. Requirements tracing on target (retro): improving software maintenance through traceability recovery. *ISSE*, 3:193–202, 2007.

[69] M. Deng, R. E. K. Stirewalt, and B. H. C. Cheng. Retrieval by construction: a traceability technique to support verification and validation of uml formalization. *ISSE*, 5:837872, 2005.

[70] F. Bouquet, E. Jaffuel, B. Legeard, F. Peureux, and M. Utting. Requirements traceability in automated test generation: application to smart card software validation. In *A-MOST '05: Proceedings of the 1st international workshop on Advances in model-based testing*, pages 1–7, New York, NY, USA, 2005. ACM. ISBN 1-59593-115-5.

[71] L. Naslavsky, H. Ziv, and D. J. Richardson. Towards traceability of model-based testing artifacts. In *A-MOST '07: Proceedings of the 3rd international workshop on Advances in model-based testing*, pages 105–114, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-850-3.

[72] L. Naslavsky and D. J. Richardson. Using traceability to support model-based regression testing. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 567–570, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-882-4.