UNIVERSITY OF CALGARY

Proactive and Reactive Decision Support for Handling Change Requests in Software Release Planning

by

Anas Ghassan Jadallah

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

April, 2010

UNIVERSITY OF
CALGARY

Abstract

Inevitable requirements volatility and dynamic changes in stakeholders' needs are two of the major threats to software projects success. This issue becomes more substantial when it comes to release planning as the most attractive features need to be assigned to the next release without violating constraints. In this research, the problem of Handling Change Requests in software release planning is addressed by a decision Support method, called Sup-HCR. Proactively, Sup-HCR provides decision support to address the modifiability concern when planning for the next release, in order to save future modification effort. Reactively, Sup-HCR provides decision support for software release re-planning, in order to accommodate change requests in the release plan. Our method includes an adapted approach for feature modeling and estimating features' impact on system modifiability, by applying object oriented design metrics to the feature domain. An initial evaluation of the methods is conducted by a real world case study.

Publications

Part of the materials, ideas, tables, and figures in this thesis have appeared previously in some of the following publications.

Refereed Conferences:

1. Anas Jadallah, Ahmed Al-Emran, Mahmood Moussavi, Guenther Ruhe, "*The How? When? and What? For the Process of Re-Planning for Product Releases*", Proceedings of the International Conference on Software Process (ICSP), Vancouver, BC, 2009, pp.24-37.
2. Anas Jadallah, Matthias Galster, Mahmood Moussavi, Guenther Ruhe, "*Balancing Value and Modifiability when Planning for the Next Release*", Proceedings of the International Conference on Software Maintenance (ICSM), Edmonton, AB, 2009. pp.495-498.
3. Ville Heikkilä, Anas Jadallah, Kristian Rautiainen, Günther Ruhe, "*Rigorous Support for Flexible Planning of Product Releases — A Stakeholder-Centric Approach and its Initial Evaluation*", Proceedings of Hawaiian International Conference on System Sciences (Mini-Track on Agile Software Development), Hawaii, 2010.
4. Ahmed Al-Emran, Anas Jadallah, Elham Paikari, Dietmar Pfhal, Guenther Ruhe, *"Application of Re-estimation in Re-planning of Software Product Releases"*, International Conference on Software Process (ICSP), Germany, 2010. (Accepted).
5. Anas Jadallah, Ira Baxter, Guenther Ruhe, "*Object Oriented Feature Modeling and its Application to Software Release Planning*", Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA), Nevada, USA, 2010 (Submitted).

Technical Reports:

1. Anas Jadallah, Matthias Galster, Mahmood Moussavi, Guenther Ruhe, "*How to Address Modifiability Concerns in the Process of Planning for the Next Release – A Trade-off Method and its Initial Evaluation*", Technical Report 084/2009, Software Engineering Decision Support Laboratory, University of Calgary, 2009.
2. Ahmed Al-Emran, Anas Jadallah, Mahmood Moussavi, Elham Paikari, Dietmar Pfahl, Guenther Ruhe, "*Functionality versus Quality: Application of Predictive Models for Re-planning of Product Releases*", Technical Report 083/2009, Software Engineering Decision Support Laboratory, University of Calgary, 2009.

Acknowledgements

All praises are due to Allah - the Almighty, the Most Gracious, and the Most Merciful - for his endless blessings upon me, for he has created me, guided me to the right path, and always been there for me. I will never be able to praise Him enough, the way he deserves to be praised.

This thesis would not have been possible without the help and support of the following people, to whom I am grateful and would like to express my gratitude:

- Dr. Guenther Ruhe, for supervision, continuous support, providing resources and research facilities, providing valuable feedback, offering direction and penetrating criticism. Without his motivation and support, I would have never made such a progress in my research.
- Dr. Armin Eberlein, for supervision, continuous support, hands-on guidance, providing valuable feedback and penetrating criticism.
- Dr. Mahmood Moussavi, for guidance, meeting and discussion, and initial effort to set up this research.
- Dr. Ira Baxter, for supporting conducting the case study, by providing all the necessary documents and valuable feedback.
- Mr. Matthias Galster, for his valuable comments, 24/7 online suggestion and inspiration.
- Mr. Ahmed Al-Emran for his valuable comments and criticism.
- All members of the Laboratory of Software Engineering Decision Support (SEDS) for providing ideas, comments, and advices to shape up my research.
- All of my family members for giving encouragement and keeping me motivated.

Dedication

This thesis is dedicated to my mother and father. My mother, for your loving care and prayer, for instilling trust in me, for teaching me early in life that knowledge is a weapon which nothing can conquer. My father, for your loving support and prayer, for being always there with me, providing remote direction and guidance. Without your support, I doubt I would have ever reached my goals.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

RP	Release Planning
HCR	The problem of Handling Change Requests
Sup-HCR	Decision Support for Handling Change Requests: the proposed method to support handling change requests
PROSUP	Proactive decisions Support for handling change requests: part of Sup-HCR method which is concerned with handling change requests before their arrival
RESUP	Reactive decisions Support for handling change requests: part of Sup-HCR method which is concerned with handling change requests after their arrival
OOFeM	Object Oriented Feature Modeling: the proposed model for representing the features in order to extract their modifiability characteristics.
FOMM	Feature Oriented Modifiability Metrics: the set of metrics adapted from the object oriented domain to estimate a feature's impact on system modifiability.
T1	Release start time
T2	Release end time
AvgNDev	Average Number of Developers
CAP	Release capacity
F	The set of features representing the existing system
f(n)	The feature with an index n in F or any other set of features.

value(n)	The value of a feature f(n) represented in a nine point scale
effort(n)	The estimated effort for implementing a feature f(n)
risk(n)	The risk of a feature f(n) represented in a nine point scale
time(n)	The arrival time of a feature f(n)
priority(n)	The priority of a feature f(n) represented in a nine point scale
S	The set of all stakeholders involved in features prioritization
F_{new}	The set of all features arriving before the start of the release period
F_b	The set of all features included in the baseline release plan
F_r	The set of all features rejected from the baseline release plan
CR(t)	The set of all new features and change requests arriving after the beginning of the release period until time t
V-THRESHOLD	Value based threshold for release re-planning
C-THRESHOLD	Cardinality based threshold for release re-planning
F_{rep}	The set of all features included in the re-planned release after performing release re-planning
R_f	The set of all features removed from the re-planned release plan F_{rep} as a result of re-planning
N_f	The set of all features replacing features in R_f as a result of re-planning
Φ	The set of all replacement between features in R_f and features in N_f as result of re-planning.
H2W	A re-planning method addresses the How? When? and What? to re-plan for software product releases

UML	Unified Modeling Language
OO	Object Orientation
TR(n)	Total number of requirements in a feature f(n)
Req(n)	The set of all requirements in a feature f(n)
TO(n)	Total number of operationalisms in a feature f(n)
Oper(n)	The set of all operationalisms in a feature f(n)
IMod(n)	The Modifiability Indicator of a feature f(n), which indicates the predicted impact of implementing f(n) on system modifiability
GQM	The Goal Question Metrics approach
FCom(n)	The Feature Complexity metric of a feature f(n)
FInh(n)	The Feature Inheritance metric of a feature f(n)
FCoup(n)	The Feature Coupling metric of a feature f(n)
FCoh(n)	The Feature Cohesion metric of a feature f(n)
Dist(n)	The distance of a feature f(n) from the ideal point
h(t)	A function which models the openness to change in software release re-planning

Chapter One: Introduction

Release planning addresses the problem of assigning features to a sequence of releases such that technical, resource, risk, and budget constraints are met [84]. However, in the process of implementing the agreed upon release plan, change requests might arrive, asking for some modifications in the plan [46]. This may cause a software project to be over budget, or even late, especially if these changes are not handled properly [98].

Therefore, the research presented in this thesis addresses the problem of change requests in software release planning. We propose a method for proactive and reactive decision support for handling change requests in this context. For proactive decision support, we are concerned with considering modifiability when planning the baseline release, i.e. before the arrival of change requests. This is assumed to save time and effort, as it helps in producing releases, containing features which are easy to modify in the future. From a reactive perspective, we are concerned with providing decision support for release re-planning process. This is assumed to produce up-to-date release plans by handling change requests once they occur and accommodate them into the release plan.

1.1 Background

In order to provide a better understanding of the research problem, a brief introduction to the key concepts of software release planning, requirements change, change management, and handling change requests is provided in the following sub-sections.

1.1.1 Software Release Planning (RP)

When developing large scale software systems with hundreds or thousands of features, software engineers usually use iterative and incremental development processes. They

adopt scheduling strategies, in which various parts of the system are developed at different times, and integrated as they are completed [47].

Following this approach for software development means that the software is not fully implemented before being delivered to the end user, but rather being released to the customer in small increments. These increments are called “releases”. Each release contains a set of features of substantial importance to the end user and the software organization. Consequently, customers receive and use parts of the system early. They can provide their feedback to the development team in order to mitigate risk. These feedback cycles make it easier to manage and maintain software projects because they allow a more flexible reaction to changes in requirements [46].

Deciding which feature to assign to which release and in what order is a complex task and often referred to as a wicked problem [69]. There are a lot of factors which impact this process: the release has pre-determined resources; there are different planning criteria such as value, urgency, frequency of use and risk; and the stakeholders have different competing priorities [67].

Software Release Planning (RP) has been recognized as being a key success factor for successful software development [84]. It is defined as the process of selecting the optimal subset of features for each release [19]. The idea is to prioritize the set of features based on different criteria. After that, features of high priority are selected for inclusion in the next release [84].

1.1.2 Requirements Change

In software development, requirements, which are the basic units for the features, tend to change frequently due to many factors, such as [46]:

- Requirements errors, conflicts, and inconsistency.
- Evolving customer/end-user knowledge of the system.
- Technical, schedule or cost problems.
- Changing customer priorities.
- Environmental changes.
- Organizational changes.

Changes in requirements can arise at any time during the development cycle. They can be in the form of addition, deletion, or modification of requirements [46].

1.1.3 Change Management

Change management is concerned with procedures, processes and standards to handle and manage changes to requirements. It must ensure that relevant information is collected and overall judgment is made about the cost and benefit of each proposed change [46]. Some change management policies cover [46]:

- The change request process and its information.
- The process of analyzing the impact, cost and traceability of change request.
- People who usually issue change requests.
- The handling process of software changes.

1.1.4 Handling Change Requests

As discussed in the previous section, handling change requests is a subset of the change management process. In the context of software release planning, the change requests are assumed to be handled by:

1. Accommodating the change requests once they occur (reactive approach)
2. Mitigating or preventing the impact of change if it occurs in the future (proactive approach)

Accommodating the change once it occurs (reactive approach) implies modifying the previously announced release plans which no longer reflect the most current needs. This can be done by adopting release re-planning processes [82]. Adopting such a process needs to be done very carefully. The change needs to be classified, assessed and prioritized in order to make informed decisions. Otherwise, we end up wasting time, money, effort and producing low quality release plans [55]. Figure 1-1 captures the re-planning process as an approach for reactive accommodation of change requests once they arrive [82].

Handling the change can also be approached from a proactive perspective. We assume that this proactive perspective requires addressing the modifiability concern at an earlier point in time, during the release planning process, which is assumed to include features with low modification effort in the release, in order to save time and effort needed when the change actually occurs.

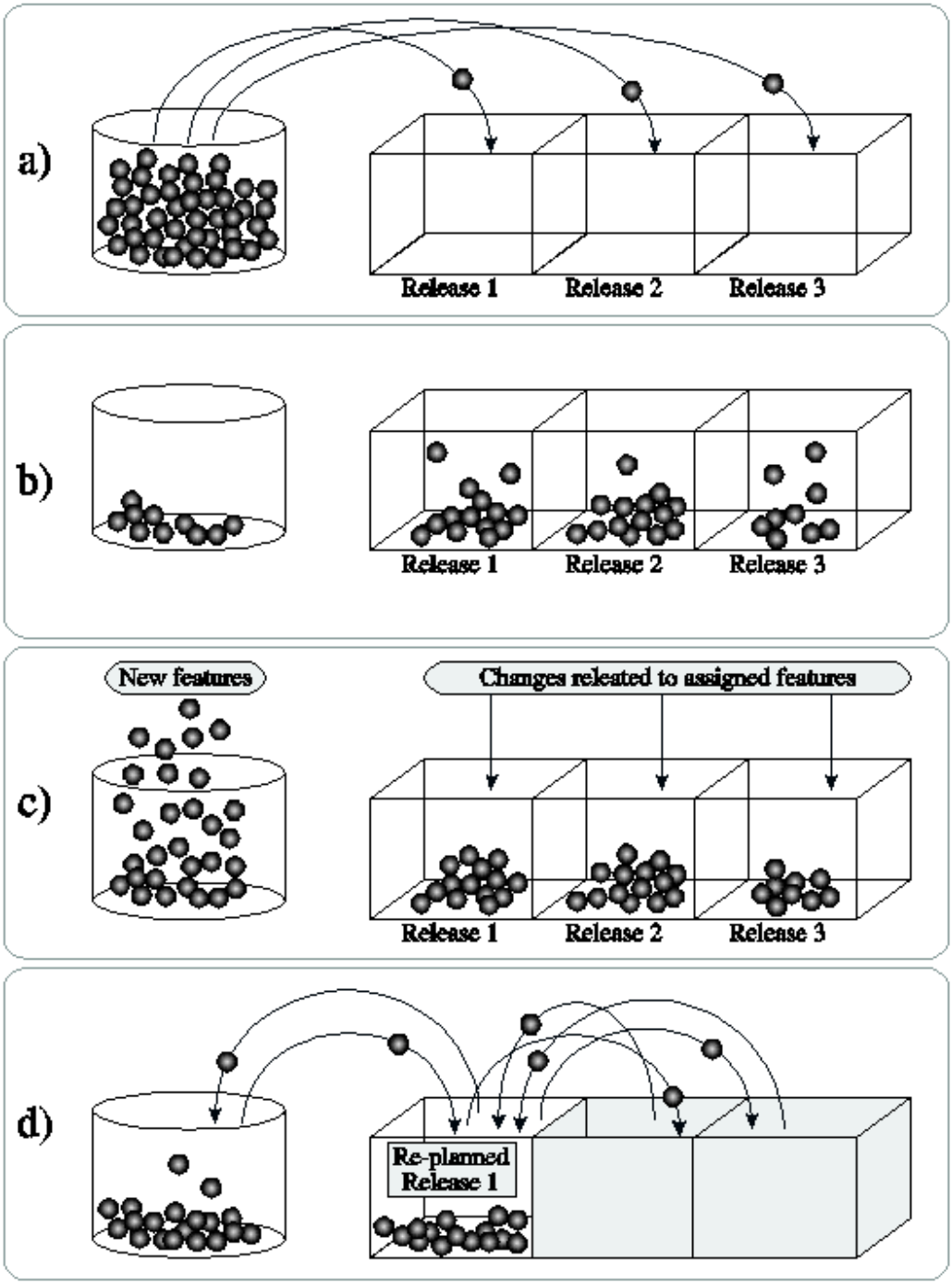

Figure 1-1: Re-planning process as an approach for reactive accommodation of change requests [82]

1.2 Overview of Research Problem

The research conducted in this thesis has the following objective:

"*Provide decision support for handling change requests in software release planning*"

This research studies the problem of change requests in software release planning. Change requests can arrive at any time after the start of activities in the release. These requests ask for changes in the announced release plan. Accommodating such changes is not a trivial task. We are looking for a method for handling these requests once they occur, as well as an approach that can reduce wasted effort of future changes in the plan.

In the following, we provide the research questions (RQ) initially stated at the start of our investigation, in order to determine the scope of our research:

RQ1: *How can we describe change requests in software release planning*?

- RQ1.1: *What kind of change requests do we expect*?
- RQ1.2: *How do change requests affect the software release*?
- RQ1.3: *What attributes can be used to characterize change requests*?

RQ2: *How can we support handling change requests in software release planning*?

- RQ2.1: *How can we provide decision support for handling change requests reactively (i.e. after the arrival of change requests)*?
- RQ2.2: *How can we provide decision support for handling change requests proactively (i.e. before the arrival of change requests)*?

Chapter 3 provides more in-depth description of the technical details of our research problem, including definitions, assumptions and research questions for each sub question of *RQ2*. *RQ1* is further discussed in Chapter 2 and 3.

1.3 Research Motivation

The problem of change is considered as one of the most substantial problems in software development. This problem is well acknowledged in the literature as one of the major causes of software project failure. A survey over 8000 projects undertaken by 350 US companies revealed that one third of the projects were never completed and one half succeeded only partially [98]. The third major cause of failure was changing requirements. Moreover, another study confirmed that there is a strong correlation between the number of change requests and the rate of software product failure [93].

The cost of changing features after delivering the product is significantly higher than the cost of addressing changes during the development process. Many studies have shown that more than 60% of the total development costs occur after the initial development has been done. More than half of the programming effort is dedicated to fixing bugs and maintaining code [71]. Other studies have shown that changes at the later stage of development have a significant impact on high severity defects affecting the system ([40] and [57]).

The problem is not with the change itself, but rather with inadequate processes for handling it. In fact, it is well acknowledged in literature that there is an increasing need for a well established process for handling change requests ([6], [94], and [90]) as well as a methodological approach for managing evolvable software releases [51]. Establishing a process for handling these requests saves time and costs and increases the likelihood of project success ([55] and [101]). This process needs not only to handle the change as soon as it occurs, but also to take proactive steps to prevent and reduce the effect of such a change should it occur in the future.

Some internationally reputable models have recognized the change management process as one of the most important key process areas for distinguishing between mature and immature organizations. For instance, the Capability Maturity Model (CMM) recognized the “change management process” as one of the key process areas at the highest level (level 5 – Optimized) of maturity [74].

1.4 Thesis Structure

The thesis is organized as follows:

- Chapter 1 presents the background of the research discussed in this thesis, the research motivation and significance, and the research problem.
- Chapter 2 presents the state of the art in the area of change requests in software release planning and their analysis, handling, and management.
- Chapter 3 states and describes our research problem in details, including key concepts definitions, assumptions and research questions that need to be answered.
- Chapter 4 provides an overview of our approach, called Sup-HCR for providing decision support for handling change requests in software release planning. It also briefly introduces its two main methods: Proactive decision Support for handling change requests (PROSUP), and Reactive decision Support for handling change requests (RESUP).
- Chapter 5 introduces an adapted approach for feature modeling and representation, called Object-Oriented Feature Modeling (OOFeM). It will be used later in Chapter 6 to estimate a feature’s impact on system modifiability.

- Chapter 6 describes our proactive method for providing decision support for handling change requests (PROSUP), including its main steps and the adapted Feature Oriented Modifiability Metrics (FOMM) in order to estimate a feature's impact on system modifiability.
- Chapter 7 describes our reactive method for providing decision support for handling change requests (RESUP), including the addressed research questions and their solution.
- Chapter 8 describes our case study, in which we applied the Sup-HCR approach to a real world project. It also provides a discussion of the outcome of the case study focusing on the strengths and limitations of Sup-HCR.
- Chapter 9 summarizes the research findings, contributions, and limitations. It also discusses directions for future research.

Chapter Two: Related Work

2.1 Introduction

The purpose of this chapter is to provide an overview of existing research conducted in the context of handling change requests in software release planning. This literature review provides the reader with an idea of how previous research addresses this problem, how different research approaches can be related to each other and how they influenced this thesis. It also provides the readers with information about problems and limitations of the previous research work.

In the next subsection, we will discuss four topics of importance to our research problem:

- Software release planning methods and techniques.
- Existing system modifications and software release planning.
- Change requests analysis in software release planning.
- Software release re-planning.

2.2 Software Release Planning Methods and Techniques

A number of known methods and techniques have been proposed for software release planning. Some are targeted towards the release planning problem while others are just requirements prioritization techniques.

In its simplest form, greedy release planning [22] creates release plans based on feature priority and resource consumption of features. It just provides guidelines on how release plans can be created by applying greedy optimization algorithms without considering how the stakeholders do the prioritization or which criteria to include in the release planning process.

Jung et al. [41] suggested a cost-value requirements analysis using mathematical programming. He applied a rigorous algorithm for solving the knapsack problem [43] to decide which requirements should be implemented into the next release.

Bagnall et al. [8] studied the problem of software release planning where the scope of planning is just the next release. They considered a model with a list of features that may depend on each other and a list of customers each with a weight and wish list. According to this method, features can only be offered if all the enabling features are provided. The goal is to maximize the total number of stakeholders being completely satisfied.

Van den Akker at al. [97] applied mathematical programming to provide a solution for the next release problem, where the main planning criterion is the projected revenue of the features.

Greer et al. proposed EVOLVE [32], an iterative approach for solving the release planning problem. The method requires the prioritization of features by the stakeholders in terms of feature value and urgency of implementation. Besides, it tries to balance conflicting stakeholder opinions to achieve the highest degree of satisfaction within given resource constraints.

An enhancement over EVOLVE was proposed recently by Ruhe et al. The newer version of the release planning method was called EVOLVE II [82]. EVOLVE II extends EVOLVE by enabling the optimization of release plans alternatives. This was achieved by integrating the ReleasePlanner^{TM} tool [3] with the release planning process. Besides, EVOLVE II provides a systematic process for release planning consisting of 13 steps. As a result of introducing new steps to the release planning, the level of stakeholders' involvement was increases, by allowing them to evaluate the set of generated optimized

release plans. This provided us with early assessment of release plans' quality from stakeholders' perspective.

None of these methods and techniques we just discussed considered the modifiability when planning for the next release, or the problem of change requests in release planning.

2.3 Existing System Modifications and Software Release Planning.

Research has been conducted to study the impact of modifying existing system components on the software release decisions. Saliu et al. developed a release planning framework by extending the hybrid intelligence method EVOLVE [89]. The framework considers historical information about the components of the system during release decision making. For this purpose, a new step called Feature-Driven Impact Analysis (FDIA) was added in order to identify system components affected by integrating new features or implementing change requests [89]. They used historical defect data to characterize the health of system components. This information is used after estimating the features' integration risk and effort to decide which features to implement in what release. Figure 2-1 provides an overview of this method.

However, this method requires a lot of information which is usually not available at the time of release planning. Also, relying only on defect data is not sufficient for characterizing the health of software components as many other factors may contribute as well, such as complexity and criticality.

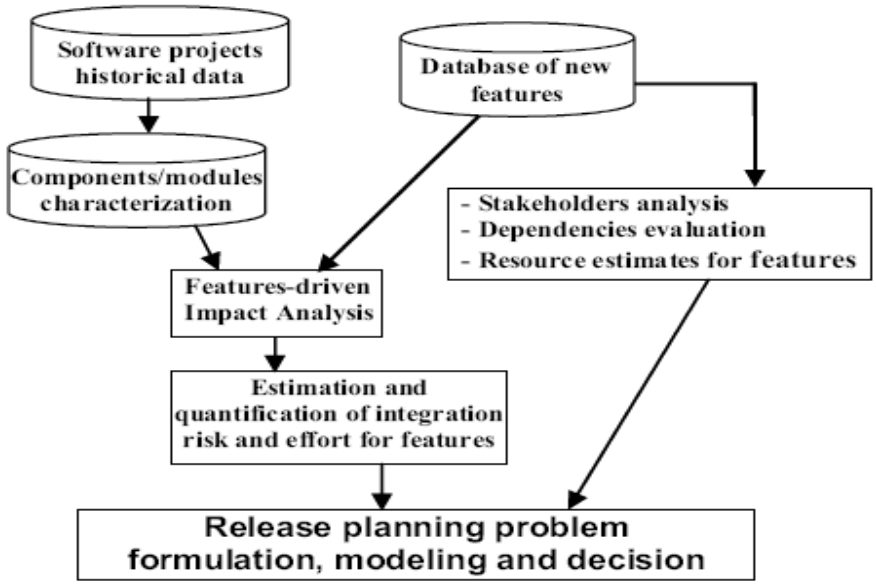

Figure 2-1: Architecture of the release planning decision support system proposed by Saliu et al. [89]

An extension to this research was presented by Saliu et al. by proposing the S-EVOLVE* method which focuses on providing more rigorous support for the release planning process for evolving systems [88]. To achieve this goal, the method considers the resources, technological constraints, and the characteristics of the target components to produce release plans which are more likely to meet the actual needs.

As part of the proposed process for estimating the characteristics of the components for the system under study, an approach for determining the potential impact of implanting new features or change requests into existing system components was developed which relies on two concepts [88]:

1. Difficulty of Modification (DoM) of a component which refers to the inherent difficulty in modifying an affected component of the system to extend or change the functionality.

2. Extent of Modification (XoM) which refers to the extent of the impact the implementation of a new feature or change request has on a particular component of the system. It is an attribute of both the feature or change request and the components hosting it.

The two concepts are illustrated in Figure 2-2. The system in the example is assumed to be composed of six components. The different grey levels of components refer to different levels of DoM. The hatched area represents the XoM.

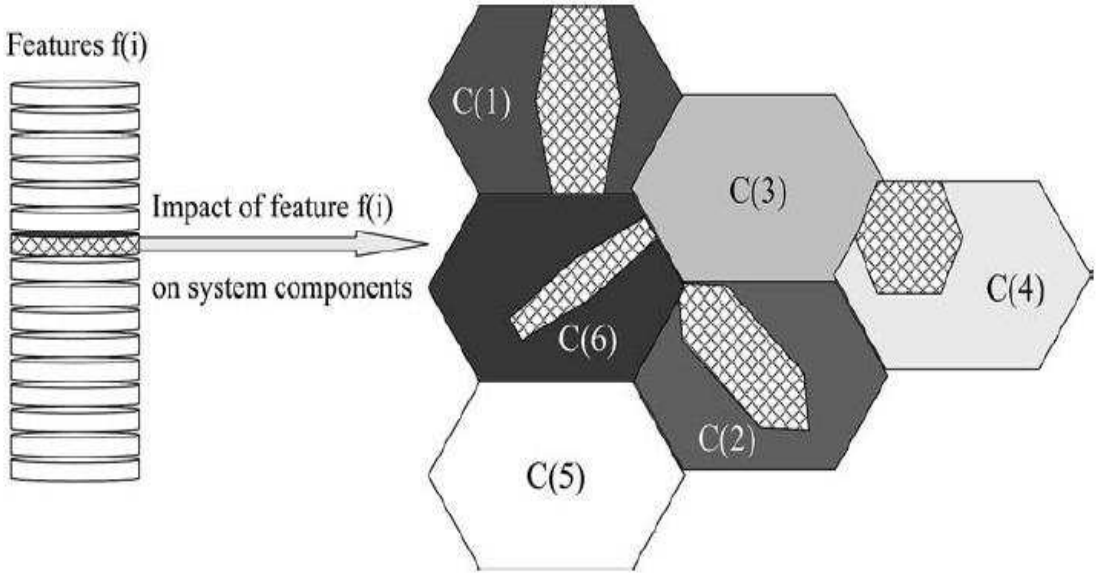

Figure 2-2: Impact of the implementation of feature f (i) on the system components C(m) [88]

A quantitative framework for deriving the weights for the substantial factors (size, complexity, understand ability, health, and criticality) was developed in order to determine the DoM. After that, pair-wise comparisons were done using AHP [86]. The XoM was assumed to be the percentage of code modification relative to the original size of the components.

However, this method again requires a lot of information at the time of release planning which may not be available. Also, the dependencies between modules and features (e.g. coupling and precedence) are not taken into consideration during the process of determining DoM and XoM.

2.4 Change Requests Analysis in Software Release Planning

Stark et al. [94] explored the relationship between requirements changes and software releases by looking at the volatility of maintenance requirements in a large software system with more than eight million lines of code spanning multiple languages, environments, and client organizations during more than 40 software releases. The research tried to answer the following questions:

- How much volatility do the product releases experience?
- What kind of change is most commonly requested?
- When in the release cycle do requirements changes most often?
- Who requests requirements changes?
- How much effort is associated with implementing each requirement type?
- What impact will a requirements change have on the schedule?

For each question, data was measured, collected, and analyzed. Finally, regression analysis was performed to find a relation between different factors affecting these change requests and the quality of release plans. A regression model was used to predict the impact of requirements changes on software releases and to facilitate communication among the project stakeholders when discussing the problem of change. It has been found that prioritizing requirements during release planning reduces modification effort for later

releases. In order to do proper requirements prioritization, three new metrics were proposed:

1. Planned schedule percentage
2. Degree of requirements volatility
3. Degree of risk

However, the study did not provide any information on how these change requests need to be handled and managed. Also, the process of re-planning an existing release plan was not studied.

Nurmuliani et al. analyzed the requirements volatility during the software development lifecycle for software delivered through a series of releases with 8000KLOC, 12-18 months of development, and 180 full-time developers per release [72]. The study proposed a quantitative method to characterize requirements changes as well as a change management process that consists of four phases:

1. Phase 1: Change requests initialization where change requests are issued.
2. Phase 2: Change requests validation and evaluation.
3. Phase 3: Change implementation.
4. Phase 4: Change verification.

They also studied the issues of change request arrival rate, requirements volatility, and taxonomies of change requests. Also, three types of change requests were suggested:

1. Adding a new requirement
2. Deleting or removing an existing requirement
3. Modifying an existing or planned requirement

Reasons for such changes are defect fixing, missing requirements, functionality enhancement, changing product strategy, design improvement, scope reduction, redundant functionality, obsolete functionality, erroneous requirements, resolving conflict, and clarifying requirements [72].

However, the research provided just an outline for how the change can be handled without a concrete description on how to prioritize or evaluate change requests (phase 2 mentioned above) and how it can be integrated/implemented within the release (phase 3 mentioned above) without affecting other requirements under development or without violating the budget and time constraints. In fact, the approach does not allow integrating all change requests. Instead a predefined percentage of change is allowed after the initial release plan is produced.

2.5 Software Release Re-planning

In the context of reactive handling of change requests, the simplest strategy is to freeze change requests [27]. This approach mandates that an organization stops accepting new change requests after starting the implementation of a release. Any changes are introduced at the start of the next release. This approach ignores the fact that some of the changes may be very urgent and need to be incorporated in the release plan immediately.

In the context of re-planning, AlBourae et al. proposed a lightweight re-planning process for software product releases that aims at producing new revised release plans with higher stakeholder satisfaction (i.e. compared to the initial baseline release plan), based on their votes, given limited time and resources [6]. In this process, change requests are accumulated, during release iterations until we reach a specified level. At this time, existing features in the baseline release plan are categorized as newly added, duplicated,

or on-going features. Also, resources are estimated for new features and stakeholders are asked to vote for their attributes (i.e. value). After that, old features are compared with new ones using AHP. Finally, modified release plans are produced based on the result of the last comparison step [6]. A process model for the proposed approach is provided in Figure 2-3.

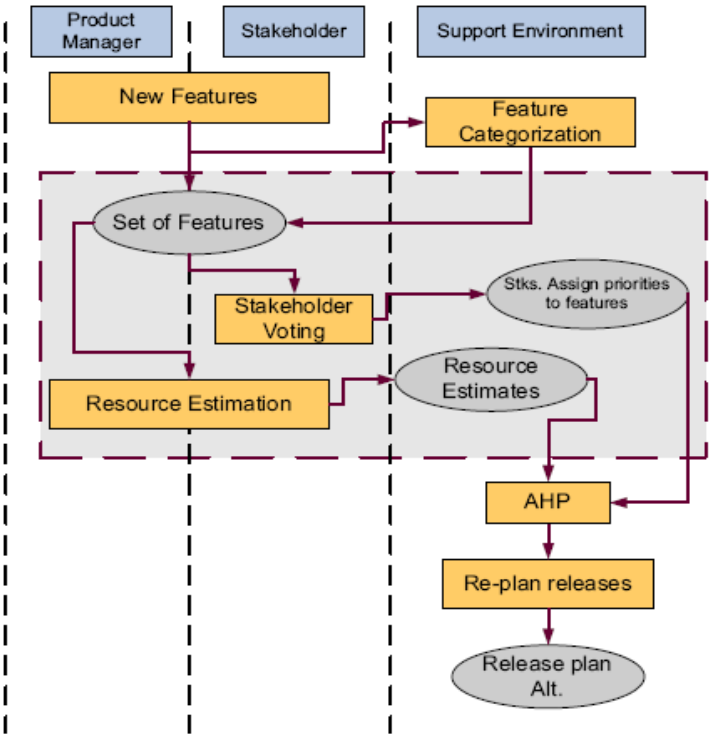

Figure 2-3: Process model for the proposed re-planning process[6]

However, the two questions of “when to re-plan?” and “what to re-plan?” were not addressed in this research.

Al-Emran et al. [5] studied the re-planning problem by focusing on discrete event simulation. They proposed a hybrid intelligence method called Planning/Re-planning (PRP) which consist of the following four steps [5]:

1. Gathering information about features to be developed and developers available for a release.
2. Initialization of the simulation model to be used.
3. Exploring the solution space by generating operational release plans using discrete event simulation.
4. Resolution where managers and experts on decision making use their knowledge and experience to address soft concerns which cannot be modeled. Also, they are asked to evaluate the different plans and finally choose one.

The method is considered to be of substantial importance in the context of release re-planning. Besides, it allows human involvement for further decision analysis and making, and it supports the revision of plans to handle unexpected changes. However, its applicability remains at the operational level of release planning.

Chapter Three: Problem Statement

In this chapter, we describe the research problem studied throughout this thesis. We start by defining the important key concepts and terminologies, including different variables and constraints affecting our investigation. After that, we describe our problem in detail, including two sub-problems as well as the assumptions and research questions which need to be answered.

3.1 Definition of Key Concepts

3.1.1 Software Release

A software release is the distribution of an initial or upgraded version of a software product to the end users [63]. This distribution is offered in the form of software features (defined below). Each release has its duration which can be determined from its start time T1 and end time T2 and resource capacity CAP which are not allowed to be exceeded.

Definition 3.1: Let AvgNDev be the average number of developers available over the release period. Then the overall release capacity CAP (in person-days) equals to:

$$CAP = (T2-T1)* AvgNDev \quad (\mathbf{3\text{-}1})$$

3.1.2 Software Feature

The concept “feature” is used throughout this thesis as the basic unit for release planning. Features are the “selling units” provided to the customer. We follow the definition given by [26] which defines the feature as “a set of logically related requirements that provide a capability to the user and enable the satisfaction of business objectives”. We assume a set of existing features $F = \{f(1), f(2), \dots, f(N)\}$ in the system under consideration.

In release planning, features are often characterized by various attributes such as value, urgency, implementation effort, risk, and frequency of use. However, in this research, we

focus on what we consider the most important attributes: value, implementation effort, and risk. These attributes have been widely used in the release planning domain ([37], [38], and [82]). They are defined below and quantified on a nine-point scale as specified in Table 3.1. They are assumed to be the results of stakeholder (and expert) evaluation as described in ([68] and [83]).

Table 3.1:Nine point scale for evaluating effort, risk and value of features

Scale	Interpretation
1	Extremely low
2	Extremely low to low
3	Low
4	Low to average
5	Average
6	Average to high
7	High
8	High to extremely high
9	Extremely high

3.1.2.1 Feature's Value

The value of a feature f(n) (denoted as value(n)) describes how valuable a feature is estimated to be to the customer or to the organization when it is released.

3.1.2.2 Feature's Implementation Effort

The implementation effort of a feature f(n) (denoted as effort(n)) describes the amount of resources estimated for implementing the feature. For estimating effort, a variety of

known methods and techniques can potentially be applied, such as Delphi [81] or Planning Poker [44]. A method for effort estimation in the context of change was studied in [77].

3.1.2.3 Feature's Risk

The risk of a feature f(n) (denoted as risk(n)) describes the relative likelihood of software failure if a feature is not released early. It is widely acknowledged in literature that there are many types of risks ([12], [66] and [61]) such as:

1. Technical risk: the potential that implementing a particular feature may waste resources. This type of risk can be determined from the bug history of the feature and the criticality of the module where the feature resides. This kind of risk is estimated by human experts involved in the development.
2. Acceptance risk: the potential that the stakeholders may not accept a particular feature if released early, because other features are more urgent for them. This kind of risk is estimated by stakeholders.

However, we will just consider the acceptance risk throughout this thesis, because we assume that the risk estimates are obtained from the stakeholders.

3.1.3 Stakeholders

A stakeholder is any party that affects or can be affected by the actions of the business as a whole [31]. Stakeholders in this thesis prioritize and vote for the feature attributes described above. An operational method for selecting stakeholders is described in [91]. We assume a given set of stakeholders $S = \{s(1),\ldots,s(q)\}$. Each stakeholder s(p) can be assigned a relative weight $\omega(p)$ (represents relative importance), based on the ordinal nine-point scale given in Table 3.1.

3.1.4 Existing Features

Existing features are the ones which have been implemented in previous releases. We assume a set of N features, F = {f(1), f(2), … , f(N)}, which represent already implemented part of the software system.

3.1.5 New Features

The new features are the ones added to the overall set of existing features before the start of the current release period. They are considered for release planning since they have not been implemented yet. We assume a set of new features, F_{new} = {f(N+1) … f(M)} are available at any time t < T1 for the next release and used for release planning. Figure 3-1 provides an example showing both the existing and new system features and their timing. The release under consideration in this example is release R3. Arrows represent the arrival of new features, while the cells represent implemented features.

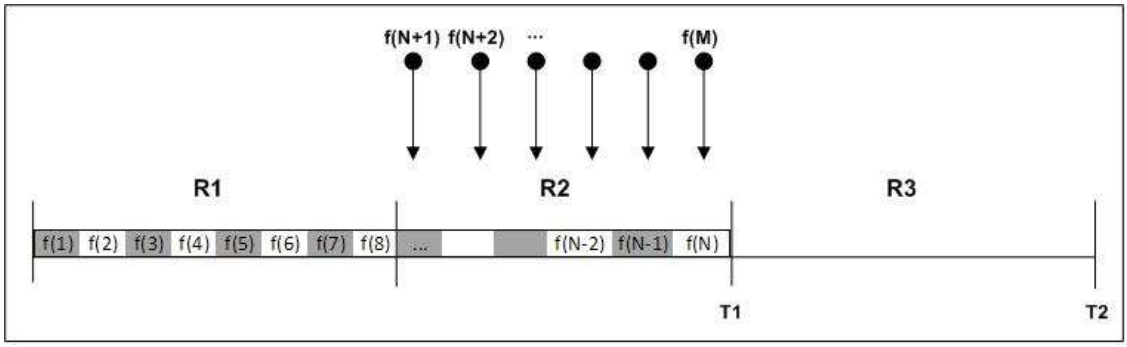

Figure 3-1: Existing and new system features

3.1.6 Baseline Release Plan

The baseline release plan represents the initial plan produced at the start of the current release (i.e. at t = T1) as a result of release planning. We assume a set of P features, F_b =

{fb(1), ..., fb(P)} selected from F_{new} for inclusion in the next release baseline plan, such that $F_b \subseteq F_{new}$.

Definition 3.2: Let F_{new} be the set of new features, CAP to be the overall capacity of the release. Then the baseline plan can be described by a Boolean vector x such that:

x(n) = 1 iff feature f(n) is selected for implementation in the next release,

x(n) = 0 otherwise (**3-2**)

($n = N+1, ..., M$)

The release capacity CAP limits the number of features that can be implemented in the baseline plan. Hence, the effort constraint becomes:

$$\Sigma_{n=N+1...M}\ x(n)\ effort(n) \leq CAP \quad (\mathbf{3\text{-}3})$$

3.1.7 Release Value

Value of a release plan x (denoted as Value(x)) is the total value of all features assigned to that release. We assume that the release plan value can be obtained by aggregating the values of the features constitute the plan.

Definition 3.3: Let x be the Boolean vector describing the baseline release planning. Then the baseline release plan value can be estimated by:

$$Value(x) = \Sigma_{n=N+1..M}\ x(n)\ value(n) \quad (\mathbf{3\text{-}4})$$

3.1.8 Rejected Features

Rejected features are the ones which are excluded from the next release plan. We assume a set of Q features, F_r = {fr(1),...,fr(Q)} are rejected from inclusion in the next release baseline plan F_b such that, $F_r \cap F_b = \Phi$, where Φ denotes the empty set, and $F_r \cup F_b = F_{new}$. The number of features in the union of F_b and F_r is equal to the number of features in F_{new} (i.e. P+Q = M-N).

Definition 3.4: Let F_{new} be the set of new features. Then the set of rejected features F_r can be described by a Boolean vector x such that:

$$x(n) = 0 \text{ iff feature } f(n) \text{ is not selected for implementation in the next release } (n = N+1..M) \quad (3\text{-}5)$$

3.1.9 Change Requests

In the process of implementing the baseline release plan, change requests might occur. A change request relates to modifying an already implemented feature, removing an already implemented feature, or requesting a new feature.

Although change requests can request new features, they differ from the set of new features mentioned above (F_{new}) in terms of their arrival time. While features in F_{new} should have arrived before the start of the release period (i.e. $t < T1$), change requests can arrive at any point in time $t \in (T1, T2]$.

Change requests as features, are characterized by value, effort and risk attributes. Besides, they are described in dependence of time as each change request can arrive at any point in time (called time(n)).

We define CR(t), the set of all change requests from the beginning of the release period ($t = T1$) until point in time $t \in (T1, T2]$. The change requests themselves are denoted by $f(M+1)$, $f(M+2)$, …. $f(M+N_t)$ where N_t denotes the number of change requests until t. Each change request f(n), $n = M+1 \ldots M+N_t$ is characterized by four attributes: value(n), risk(n), time(n) and effort(n). Figure 3-2 provides an example showing change requests and their arrival after starting the baseline release plan implementation.

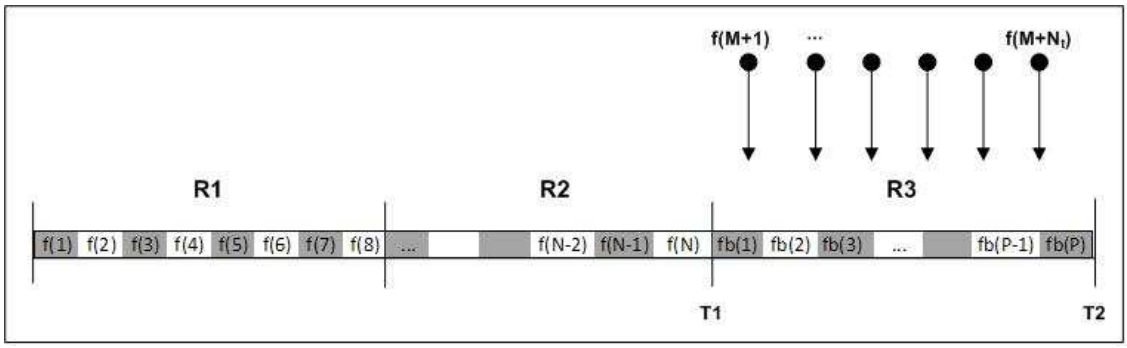

Figure 3-2: Arrival of change requests

3.1.10 Change Threshold

The change threshold represents the degree of change in the baseline release plan at which there is enough reason to update the plan with new features and change requests. Change threshold can be identified for any attribute characterising a feature such as: value, effort, and risk. For example, the value-based threshold (V-THRESHOLD) study the accumulated value of arriving change requests, and make sure that these requests are accommodated in the plan whenever a certain threshold value is reached; while the cardinality-based threshold (C-THRESHOLD) study the accumulated number of arriving change requests, and make sure that these requests are accommodated in the plan whenever a certain number of them has arrived.

3.1.11 Re-planned Release

Any accepted change request causes the baseline release plan to be no longer up-to-date. For this reason, software release re-planning is performed. Hence, a re-planned release represents the modified release plan produced from the baseline release plan, the set of rejected features and the change requests that arrived before the time of re-planning. We assume a set of R features, F_{rep} = {frp(1), ..., frp(R)}are selected from F_{new} and CR(t) for

inclusion in the re-planned release as a result of re-planning, such that $F_{rep} \subseteq (F_{new} \cup CR(t))$.

Definition 3.5: Let F_{new} be the set of new features that arrived before T1, CR(t) be the set of all change requests that arrived until time $t \in (T1, T2]$. Then the re-planned release at time t can be described by a Boolean vector xr such that:

xr(n,t) = 1 iff feature f(n) has been implemented before t or if it is selected for implementation in the re-planned release (**3-6**)

$$(n = N+1..N_t)$$

Hence, the features considered in the effort constraint become:

$$\Sigma_{n=N+1...Nt}\ x(n)\ effort(n) \leq CAP \quad (\textbf{3-7})$$

3.1.12 Release Stability

Once a release plan is announced to the end users, it cannot be completely changed if new features arise. Instead, only a particular percentage of change is allowed to keep the release stable as much as possible. Hence, the degree of release stability is calculated from the degree of change in the baseline release plan. It can be estimated from the percentage of number of features not replaced (after re-planning) to the maximum number of replacements in case all features in F_b are replaced during re-planning.

Definition 3.6: Let F_b be the set of features in baseline release plan, F_{rep} be the set of features in the re-planned release. Then the stability of the baseline release plan can be estimated by:

$$Stability = Card(F_b \cap F_{rep}) / Card(F_b) \quad (\textbf{3-8})$$

3.1.13 Feature Dependencies

Feature dependencies are used throughout this thesis to represent the technical constraints in sequencing and grouping features in different releases.

We model what we consider to be the two most important types of feature dependencies: coupling C and the precedence P dependencies, which are widely acknowledged in the context of release planning ([84] and [65]). Our definition of coupling and precedence in our context, differs to some extent from the traditional definition in literature [82]. It can be summarized as follows:

- Coupling: Feature f(i) is coupled to feature f(n) if f(i) depends on part of f(n) for its operation. This dependency is due to implementation issues (i.e. functionality of a particular feature cannot work without another feature's functionality). In this case, f(n) should be assigned to the same or earlier release as f(i) but not vice versa.
- Precedence: Two features have precedence dependency if one feature can only be implemented if the other feature has already been implemented. This dependency is due to the fact that a feature f(i) needs the whole feature f(n) to be reused in its own implementation. In this case, f(n) precedes f(i), and then f(n) needs to be implemented before f(i). This can be in the same release or in different releases.

3.1.14 Impact on Modifiability

Modifiability is defined as "the ease with which a software system can accommodate changes" [7]. We use the term "impact on modifiability" throughout this thesis, to express the degree to which the implementation of a particular feature is assumed to

affect the overall system modifiability, which indicates to some extent the degree to which this feature is designed to accommodate future change requests.

3.2 Problem Description

Our research problem, called HCR is concerned with Handling Change Requests in release planning. It is conducted in the context of planning and re-planning for the next release.

The HCR problem is based on different types of information related to existing system features, baseline release plan, change requests, resource capacities and time interval for the release under investigation.

The HCR problem starts at t = T1. At this point in time, change requests have not yet arrived. On the other hand, features in F_{new} are assumed to have arrived and ready to be used in the baseline release planning. However, features in F_{new} may have dependencies with each other and with existing system features in F, such as coupling and precedence dependencies. These dependencies as well as the structural properties of features (i.e. complexity) may affect any change request handling strategy in the future. So, our first part of the problem, called HCR-Pro is stated. It is concerned with considering features dependencies and structural properties in order to include easy-to-modify (modifiable) features in the next release plan. This is assumed to save modification effort when change requests arrive in the future.

The second part of the problem, called HCR-Re is concerned with handling change requests after their arrival. The HCR-Re problem starts at time $t \in (T1, T2]$. Change requests have accumulated during the release period until this point in time, when the change threshold is exceeded. This provides enough reason to update the baseline plan by

re-planning. HCR-Re is concerned with addressing change requests after their arrival by performing release re-planning.

The next two subsections discuss the two problems, HCR-Pro and HCR-Re in more detail.

3.2.1 Problem 1: Proactive Handling of Change Requests HCR-Pro

The term “proactive” is used to express any action taken in anticipation of future problems, needs, or changes [58]. Hence, proactive handling of change requests is concerned with acting up-front, before the arrival of change requests, in order to increase the likelihood of handling them when they arrive.

HCR-Pro is concerned with addressing modifiability during the release planning process. Consequently, HCR-Pro is assumed to increase the system modifiability and to simplify the implementation of change requests when they arrive. It is based on the following assumptions:

1. Planning is only done for the next release.
2. The features considered for baseline release planning are the ones in F_{new}.
3. Value, effort, and risk estimates are available for all features in F_{new}.
4. Features in F_{new} may have dependencies with each other and with existing system features in F.
5. Structural properties of features and their dependencies affect accommodating future change requests.

With the HCR-Pro setup described above, four key decisions need to be made, which are:

1. What are the relevant attributes from OO domain, which can be mapped to features in release planning domain in order to estimate a feature's impact on system modifiability?
2. How can these attributes be extracted from the features?
3. How can these attributes be used to indicate a feature's impact on system modifiability?
4. How can a feature's impact on system modifiability be considered in the release planning process?

3.2.2 Problem 2: Reactive Handling of Change Requests HCR-Re

The term "reactive" is used to express any action taken as a response to a stimulus. HCR-Re is concerned with release re-planning as a reactive approach for handling change requests once they arrive. It is based on the following assumptions:

1. Re-planning is only done for the next release.
2. The features considered for re- planning are as follows:
    a. The set of features in the baseline release plan (F_b).
    b. The set of rejected features in F_r
    c. The set of change requests in CR(t)
3. Value, effort, and risk estimates are available for all features in F_b, F_r and CR(t).
4. Features in F_b, F_r and CR(t) may have coupling and precedence dependencies.
5. The start time start(n) and end time end(n) of features implementation in F_b are given as a result of operational planning for the baseline release plan.
6. It is possible to estimate the resource available until the end of the release (i.e. t = T2).

With the HCR-Re setup described above, four key decisions need to be made, which are:

1. When to re-plan? The question has to be asked is: at what point in time the accumulation of change requests has provided decision maker with enough reason to change the baseline plan.
2. Which features to re-plan? This question refers to the decisions which need to be taken regarding including features in the baseline release plan in re-planning. This is because at the re-planning time, some features in F_b have already been implemented; some other features are scheduled to be implemented; while others are currently being implemented.
3. How to re-plan? This question refers to the procedure of prioritizing features and generating new up-to-date release plans.
4. What features to keep after re-planning? This question refers to the trade-off between the release stability and the added value of new features after re-planning. A balance has to be found between instability and added value caused by the replacement of features.

Chapter Four: Decision Support for Handling Change Requests

In this chapter, we briefly introduce our approach, called Sup-HCR for supporting change requests handling in software release planning. This approach is further explained in detail in the subsequent chapters.

Sup-HCR supports handling change requests from two different perspectives: proactive and reactive. Proactively, it starts by addressing HCR-Pro problem at the earliest point in time, before the arrival of change requests, when baseline release planning is performed. The first part of Sup-HCR, called PROSUP, tries to include easy-to-modify features in the baseline release plan by addressing the modifiability concern in release planning ([38] and [39]). The assumption is that adding another criterion to release planning that considers the impact of features on system modifiability will save effort in the future, when change requests arrive. Hence, PROSUP is mainly concerned with modeling and estimating the impact of features on system modifiability.

Reactively, the second part of Sup-HCR, called RESUP supports handling change requests after their arrival by addressing the HCR-Re problem. For this purpose, it uses the H2W [37] method. It starts by determining when to start re-planning. Then, it selects the subset of features to include in the re-planning. After that it produces a new up-to-date release plan by applying greedy heuristics and prioritizing features according to their distance from the ideal point. Finally, it determines the number of feature replacements by compromising between the additional value of change requests, and the allowed degree of change in the release.

Figure 4-1 shows the high level design of Sup-HCR. It includes the following steps:

1. Step 1: New features are elicited and used for the baseline release planning. Part of these features represents the basic units for the baseline release plans. These features are usually elicited by requirement engineers.
2. Step 2: Stakeholders vote for features' value, effort and risk. These votes play a role in features prioritization when performing baseline release planning.
3. Step 3: The impact of features on system modifiability is estimated. Considering this attribute in features prioritization during release planning is assumed to simplify accommodating future change requests. It is estimated as follows:
    a. Features are represented in such a way that allows us to extract modifiability information such as complexity, coupling and precedence dependencies, as well as cohesion. This representation is called Object Oriented Feature Modeling (OOFeM) and further discussed in Chapter 5.
    b. Features modifiability metrics are extracted from the OOFeM, and aggregated in to "a feature's impact on modifiability" attribute. This attribute is the one used in release planning to address the modifiability concern.
4. Step 4: Release planning is performed for the next release considering features value, effort, risk and impact on modifiability.
5. Step 5: After the start of the release period, change requests arrive and accumulate at different points in time. These change requests make the current baseline release plan no longer up-to-date.
6. Step 6: Release re-planning is performed in order to support reactive handling of change requests. It uses the H2W method as follows:

a. Specifying the re-planning time. This time at which enough change requests have accumulated to justify a change in the baseline plan.
b. Selecting the set of features to consider for re-planning. This set of features includes change requests, rejected features in baseline release planning, and baseline release plan features after evaluating their status at the time of re-planning.
c. Prioritizing features according to their distance from an "ideal" point that represents the best compromise between features value, effort, risk and impact on modifiability.
d. Generating a new release plan by applying greedy heuristics on the prioritized features.
e. Adjusting the new release plan after compromising between the additional value of change requests and the allowed degree of change in the baseline release plan. This compromise determines the best number of feature-replacements.

7. Step 7: Re-planning results in generating new up-to-date release plan accommodating new change requests.
8. Step 8: The set of system features is updated based on the output of the re-planning process.

Proactive support for handling change requests (PROSUP) is mainly concerned with Step 3 and discussed in details in Chapter 6, while Reactive support for handling change requests (RESUP) is concerned with Step 6 and discussed in details in Chapter 7

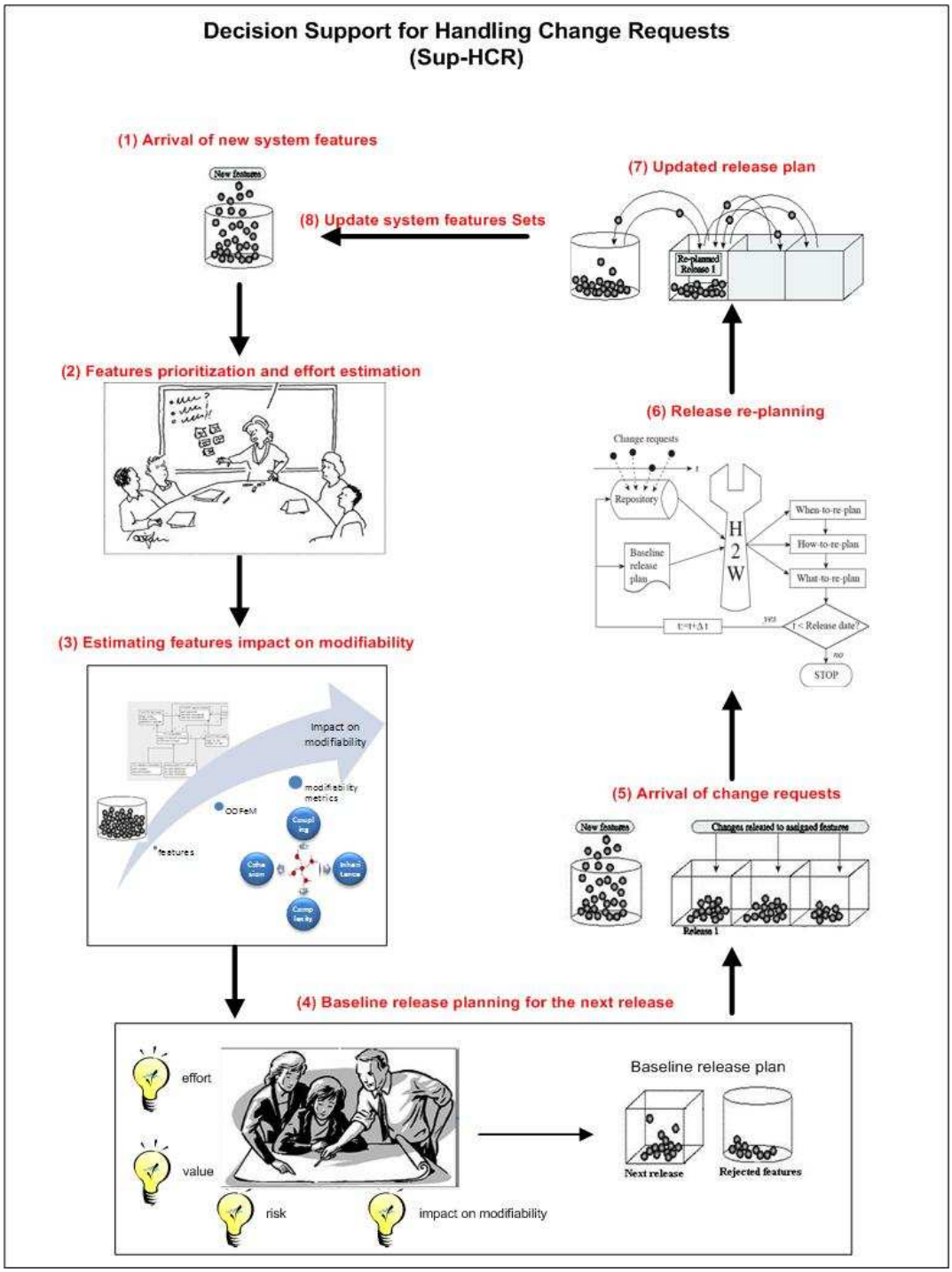

Figure 4-1: Overview of Sup-HCR

Chapter Five: Object Oriented Feature Modeling

5.1 Introduction

As discussed in Chapter 4, one of the most important steps in the proactive decision support for handling change requests (PROSUP) is to address the modifiability concern in release planning ([38] and [39]), which can be achieved by estimating the impact of features on modifiability, and integrating it in the release planning decision making. In order to estimate the impact on modifiability, features need to be modeled such that it is possible to extract modifiability information.

In this chapter, we describe an approach for feature modeling adapted from other disciplines to the release planning domain. The approach uses concepts from object orientation [45] and maps them to features in release planning, in order to capture the intra- and inter-structural characteristics of features (i.e. coupling and cohesion). We adapted this approach because none of the existing feature models (see Section 5.2) can be used for the estimation of the features' impact on system modifiability.

The proposed Object-Oriented Feature Modeling (OOFeM) utilizes concepts such as complexity, coupling, inheritance and cohesion, known from UML structure diagrams, which model the static structure of a system ([62], [59]). In contrast, behavioural diagrams (such as use-case diagrams or sequence diagrams) show the dynamic behaviour between objects. In particular, we utilize the concept of class diagrams as one particular type of structure diagrams. The motivation behind translating features and their dependencies into a UML class-diagram-like model is that the structural characteristics of features in release planning resemble to some extent the classes in UML diagrams. This similarity can be summarized as follows:

- Both classes in UML and features in release planning can be characterized by their complexity.
- As classes in UML have coupling dependencies due to functionality reasons, features in release planning can be coupled together to provide certain functionality.
- As classes in UML have inheritance hierarchies, in which one class cannot be implemented unless it reuses another class, features in release planning have precedence dependency, in which one feature cannot be implemented before another one.
- As classes in UML represent modules with highly cohesive properties, features in release planning represent collections of highly cohesive requirements.

So, we intend to utilize structural properties of class diagrams, measured in OO design metrics [21]. These metrics have been widely used for estimating the modifiability of software systems ([18], [25], [54]). This translation allows us to apply object-oriented design metrics to estimate the impact of a feature on system modifiability, as will be discussed in Chapter 6.

5.2 Related Work

Feature modeling has traditionally been used for domain analysis in software product lines. It captures stakeholder-visible aspects of software systems [30]. Such aspects not only refer to the end user, but could be of interest for any stakeholder, such as developers [49]. Feature models hierarchically structure features and define common and variable characteristics in product lines. These models also describe relations between features [49], [78], [79].

Even though conventional feature models for product lines model dependencies between features, they do not investigate modifiability based on these dependencies. Also, the internal structure of features is not considered from a modifiability perspective. Fey et al. [30] used inter-feature dependencies to model commonalities and variability in product lines. Similarly, Ferber et al. investigate feature dependencies in product lines to derive variants from legacy product line assets [29]. In both studies, dependencies have not been used in order to assess the degree of modifiability of products.

Czarnecki et al. [24] studied the relationship between feature modeling and ontology modeling. Ontology represents cohesive organization of hierarchal domain knowledge, while features in product lines view hierarchal structure of system requirements. Hence, feature was mapped to ontology. This hierarchical structure mapping was studied from semantics and integration perspectives.

UML multiplicities, which represent associations between classes in UML class diagrams, have been applied in product line feature models by Riebisch et al.[79]. As multiplicities related to dependencies are less obvious in feature models and sometimes difficult to express, a change using UML in the notation of features was suggested. This lack of semantics in feature models has also been discussed by Bontemps [13].

UML class notations has been utilized by Turner et al. [96] who tried to develop a framework for feature engineering. They studied the role of features within processes and life-cycle activities. Based on their study, many definitions for features were suggested, which relate features to system requirements, code, and lifecycle artefacts. They utilized class diagrams in their case study to model features, which were studied at the artefacts level. Subsets of lifecycle artefacts were mapped to features which were mapped to

classes. However, the internal structure of the class was not studies. Besides, the association between classes was used to describe the relations between system artefacts. The goal of the study was to develop basic concepts and terminologies that can be used as a foundation for a framework of feature engineering.

UML has also been utilized by Griss et al. [33] who used the reuse-driven software engineering business (RSEB), a model-driven approach to software reuse based on object-oriented software engineering. RSEB utilizes UML to model systems in the product-line domain. Classes from UML are redefined as features; however, this class-feature mapping was studied from a reuse perspective. Also, the internal structure of classes was not adapted to the feature domain.

Botterweck et al. [14] proposed EvoFM, a feature model for product line evolution. This model captures changes in features' structure and dependencies, to assist in the "proactive strategic planning". However, the intention of EvoFM was to investigate product line evolution.

5.3 Features as Classes

A class notation in UML consists of three sections: class name, class state or attributes, and class methods ([73] and [62]). In OOFeM the class name is simply mapped to the feature name or feature identifier.

Similarly, we map software system requirements to important properties of software systems, which must be preserved in order to provide the desired functionality. For example, the test coverage tool which is presented in the example of Section 5.5 has the following requirement:

R23 "The user must be able to view the summarized test coverage data including the total number of probes, total number of files, percentage of coverage code and the percentage of uncovered code"

This requirement is transferred to the system property:

P14 "Summarized test coverage data".

Once requirements are mapped to properties, and since a feature is defined as a set of logically related requirements [26] as described in Section 3.1.2, software system requirements can easily substitute the attributes in class diagrams. This is because both requirements and attributes describe system's properties.

Transferring requirements to properties, results in losing information from requirements specifications. For example, transferring requirement R23 in the previous example to the property P14 resulted in losing additional information related to the content of summarized test coverage data, such as total number of probes and files as well as the percentage of covered and uncovered code. We substitute for this loss of information by adding another level of detail when we map the methods of the class. The assumption is that we need to add more operational information about requirements from the specifications to tell us how requirements can be addressed.

Hence, the methods of a class, which describe the dynamic behaviour of classes in UML, are mapped into (testable) operationalisms. Operationalisms are entities that make requirements operational and enable the feature to meet the requirements [100].

Operationalisms are specific to features and cannot occur in other features or spread across features, except in an inheritance hierarchy.

Operationalisms always address particular requirements in a feature and ensure that the requirements are preserved. For example, the following operationalisms were added to describe the property P14 “Summarized test coverage data” in order to describe operationally how it can be addressed.

1. O14.2.1 “Show total number of probes”
2. O14.2.2 “Show total number of files”
3. O14.2.3 “Show percentage of covered code”
4. O14.2.4 “Show percentage of uncovered code”

A graphical representation of a feature and its relation to a class is shown in Figure 5-1.

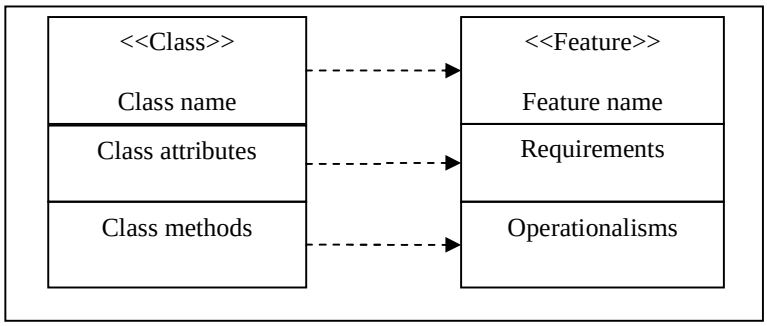

Figure 5-1: Feature-Class mapping in OOFeM

Definition 5.1: Let TR(n) be the total number of requirement in feature f(n), Req(n) = {r(n,1), r(n,2), ..., r(n,TR(n))} be the set of requirements of feature f(n). Then there exists a set of TO(n) operationalisms Oper(n) = {o(n,1), o(n,2), ..., o(n,TO(n))} for the feature f(n) such that:

$$\text{isAddr}(o(n,i), r(n,j)) = 1 \text{ iff operationalism } o(n,i) \text{ satisfies requirement } r(n,j) \text{ of feature } f(n) \quad (5\text{-}1)$$

5.4 Interaction between Features in OOFeM

OOFeM implements two types of feature dependencies: coupling and precedence. Coupling between features occurs when one feature requires certain functionality from another feature. Precedence dependency occurs when one feature cannot be implemented unless another feature is completely implemented. These dependencies were described in detail in Section 3.1.13.

Similar as with features and classes, these dependencies are related to dependencies in UML class diagrams [73] as specified in Table 5-1. For the notation of dependencies in UML we refer to [59].

Table 5-1: Feature dependencies

Feature dependency	Notation	UML class diagram artefact
Coupling	<couples>	Association
Precedes	<precedes>	Inheritance

5.5 OOFeM Example

In this section, we present a simple example to show the usefulness of OOFeM. The features modeled in this example are taken during the release planning of a tool called "C# 3.0 Test Coverage Tool" [23]. This example is part of the case study presented in Chapter 8.

The OOFeM is shown in Figure 5-2. This model was developed by requirements engineers after analysing the following documents:

1. Tool documentation
2. Requirements documents

3. Tool functionalities

The model shows many types of dependencies: f(7) “Source Display Window” is coupled to f(6) “Fast Source File Reader”, because f(6) reads the source code file to be displayed by f(7). In the same way, f(14) “Action Panel Report Generation” is coupled to both f(11) “Action Panel TCV Arithmetic”, and f(13) “TCV Statistics Computation”, because f(14) generates reports summarizing the test coverage arithmetic results provided by f(11), as well as the test coverage statistics results provided by f(13).

Features f(11) “Action Panel TCV Arithmetic” extends f(10) “Basic TCV Computation”. This is because f(11) reuses the functionality provided by f(10) and adds arithmetic operations to it in order to find the difference, intersection and union of test coverage computations.

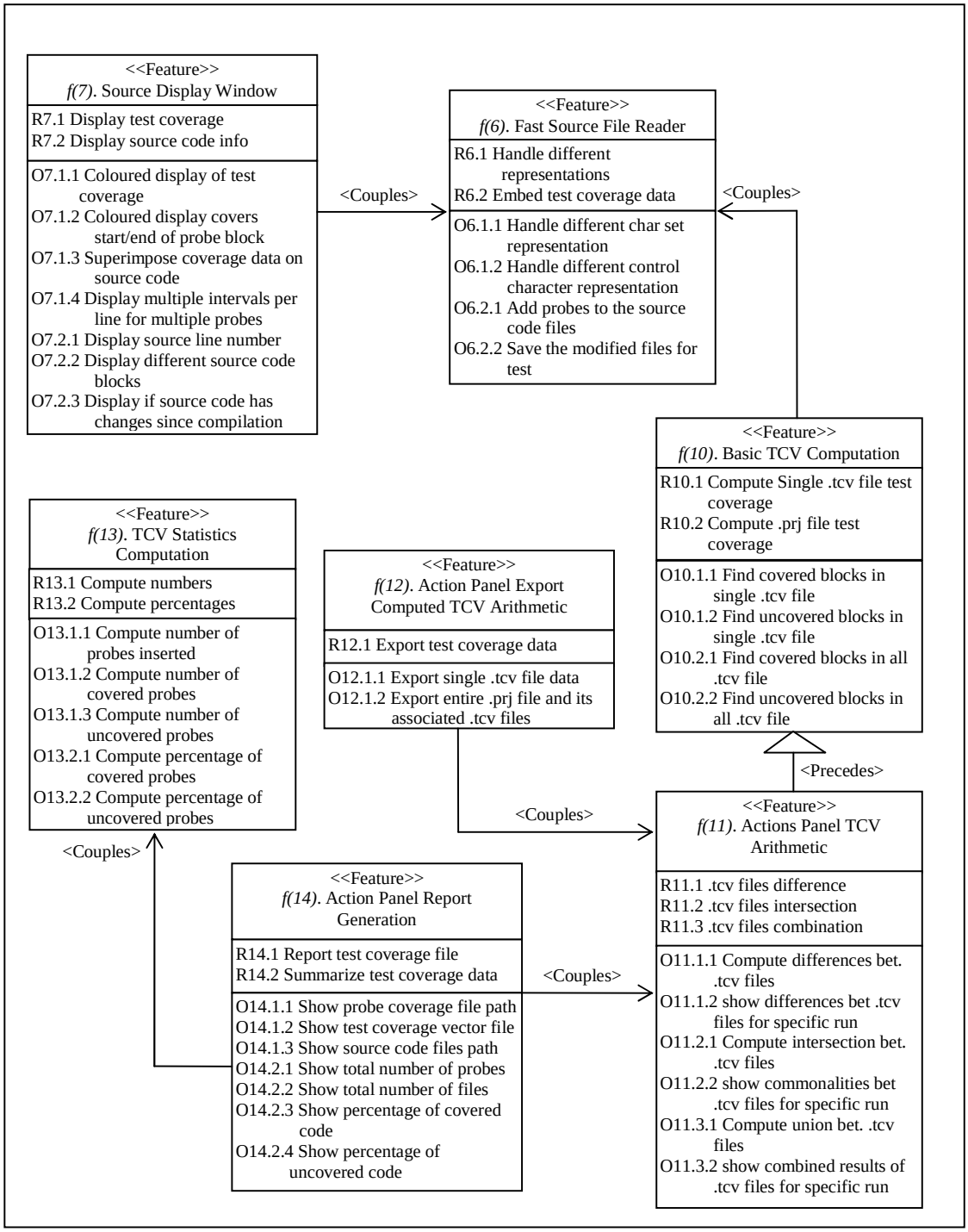

Figure 5-2: OOFeM for CSharp 3.0 Test Coverage Tool

Chapter Six: Proactive Decision Support for Handling Change Requests (PROSUP)

In this chapter, we present the first part of our approach, which deals with proactive decision support for handling change requests (PROSUP). PROSUP addresses the modifiability concern during baseline release planning, before the arrival of change requests. It supports including features which are easy-to-modify in the baseline release plan, which is assumed to save future modification effort when the change requests arrive. We start by providing an overview of PROSUP, including its assumptions and main steps. After that, we discuss in more detail the technical aspects of each step.

6.1 Overview

The main motivation behind PROSUP is to reduce modification effort in future. For this purpose, we aim at designing releases, containing features which are easy-to-modify. This is assumed to be achieved by addressing the modifiability concern in the baseline release planning. We suggest adding another criterion for release planning at the feature level, called “impact on modifiability” besides the current criteria (i.e. value, effort, and risk). In order to approximate a feature’s impact on system modifiability, we assume the following:

1. Features to be released may depend on each other and on already existing system features in F.
2. Structural properties of features (i.e. complexity and cohesion) as well as their dependencies (i.e coupling and precedence) affect future change requests.
3. Addressing the modifiability concern in release planning at an early point in time (i.e. $t = T1$) saves effort in accommodating future change requests (i.e. $t \in (T1, T2]$).

Figure 6-1 provides an overview of our approach, which consists of the following steps:

1. Step 1: Model system features using object-oriented feature modeling (OOFeM). Visualizing the structural properties and dependencies between features using OOFeM allows us to identify their salient characteristics and extract their modifiability information.
2. Step 2: Extract the feature-oriented modifiability metrics (FOMM) from the produced OOFeM in Step1 such as: complexity, coupling, inheritance and cohesion metrics. These metrics allow us to estimate a feature's impact on system modifiability in the next step.
3. Step 3: Aggregate these modifiability metrics (FOMM) into the features' modifiability indicator (IMod), which represents the estimated impact of implementing particular feature on system modifiability. Once this modifiability indicator is estimated, it can be easily transferred to the nine point scale and eventually used in release planning as another attribute of features.

We will discuss these steps in details in the following sections. We will also include a description of how feature impact on modifiability is defined and estimated.

Before explaining the technical details of each step, we need to map the term "modifiability" to our domain. Hence, the next section provides a modifiability approximation for the software release planning domain. It discusses the following questions:

1. How can system modifiability be estimated from features that constitute the system?

2. Which one affects estimating system modifiability more? Internal modifiability characteristics of features that constitute the system or their dependency on other existing features in the software systems?

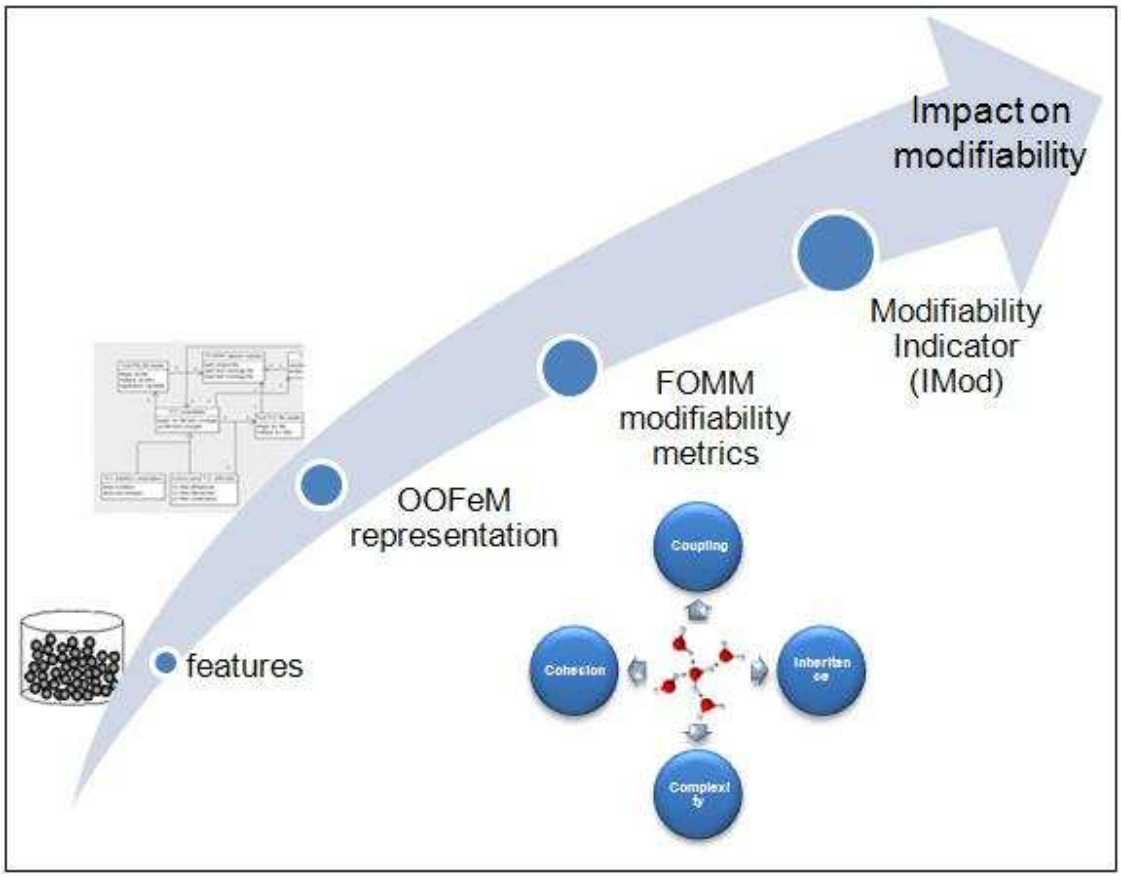

Figure 6-1: Overview of PROSUP

6.2 Modifiability Approximation

The term "modifiability" is acknowledged in literature as being related to the "cost of change" ([11], [60], [70] and [76]). In this thesis, we follow the definition given by the Software Engineering Institute (SEI), which defines modifiability as: "the ease with which a software system can accommodate changes" [7].

Several researchers have characterized modifiability of software systems as a system attribute while others considered it as a module attribute. In this thesis, we follow the definition given by [76] which considers software system modifiability neither as a system nor as a module characteristic. Instead, it considers modifiability as a compromise between both. The compromise is done by estimating module modifiability from its internal modifiability characteristics at first. On the other hand, modifiability is highly affected by the strength of interaction between this module and the overall environment (software system). So, this modular modifiability measure is readjusted at the next step by considering the environment in which this module resides. Finally, the software system modifiability is estimated by aggregating the modifiability measures of all modules that constitute the system.

We treated modifiability in the context of software release planning in the same manner.

An analogy to modules in software systems, features represent the basic units of the software system. So, we assume that as software system modifiability can be estimated from its modules, it can also be estimated from the features making up the system. We start by estimating a feature's impact on modifiability from its internal modifiability characteristics as described above in estimating module modifiability. We capture these internal modifiability characteristics using a feature's internal complexity and cohesion. On the other hand, features may have dependencies with other features in the system. These dependencies affect the overall system modifiability. So, we combine this external modifiability concern with the internal one in order to estimate the features' impact on system modifiability. We capture a particular feature's external modifiability characteristics form the strength of interaction between this feature and the other features

in the system. This interaction can be in the form of coupling and precedence. Addressing both a feature's internal and external modifiability characteristic enables us to estimate features modifiability indicator (IMod) which is assumed to indicate the feature's impact on system modifiability.

The use of the concepts of "module" and "feature" to approximate system modifiability does not violate our previous mapping between "features" and "classes" as described in OOFeM in Chapter 5. This is because these two concepts were used just to identify factors affecting modifiability based on the assumption that features of a software system resemble to some extent the modules as the software system is the big block of smaller units (features or modules).

6.3 Related Work

Since we are using OO to represent features and extract their OO modifiability metrics, this section discusses the OO modifiability metrics, and the related work on modifiability from release planning perspective. This review of related research helps us to determine the most appropriate metrics to estimate a feature's impact on modifiability as discussed in Section 6.4.

6.3.1 Features Dependencies and Modifiability in Software Release Planning

Feature dependency has been studied from different perspectives by Carlshamre et al. [20] and Saliu et al. [87]. Carlshamre et al. studied these dependencies at the granularity of requirements composing the features, while Saliu et al. considered it at the feature level.

Carlshamre et al. conducted a survey of five different companies from different market segments to study requirements dependencies in software release planning ([20] and [19]). He found that:

- Only 20% of requirements were singular. This means that most requirements depend on other requirements.
- 20% of requirements were responsible for 75% of dependencies. These requirements need to be handled carefully.
- Identifying independent requirements in earlier releases reduces modification effort at a later stage.
- Visualization of requirements dependencies is efficient in supporting the identification of their salient characteristics. This is because requirements are usually fuzzy and their sheer number makes them difficult to identify and manage.

Based on the results of this survey, different types of requirements dependencies were identified. Coupling, precedence, and exclusion appear to be the most important ones.

Carlshamre et al. also proposed a dependency measure in order to evaluate different release plan alternatives. This measure is called “release coupling” and indicates to what extent the requirements within a release are coupled with each other. It relates the coupling between requirements in the release planning domain with coupling of objects in OO design: In both cases a high degree of coupling indicates higher sensitivity to change in other parts of the system. Also, changes to highly dependent requirements are most likely to affect many requirements ([20] and [19]).

Saliu et al. studied the problem of software release planning from a bi-objective perspective. The study tries to optimize the business and implementation values of release plans. For the implementation value, the research focused on facilitating reuse and saving development resources [87]. The proposed method, called Bi-Objective Release Planning for Evolving Systems (BORPES), quantifies coupling between features based on the overlap in their implementation and integrates this information in the release planning decision making process. Finally, it generates alternative plans to address decision making under uncertainty. Only one kind of dependency (i.e. coupling) was studied though.

6.3.2 Object Oriented Design Metrics and Modifiability

It is widely acknowledged in literature that object-oriented design metrics can help to predict system modifiability and maintainability. Li et al. collected maintenance effort data from two commercial systems over three years [54] and found that maintenance effort can be predicted based on the MOOD set of metrics, a set of object-oriented (OO) design metrics [18]. This set includes: Depth-of-Inheritance Tree (DIT), Number of Children (NOC), Coupling between Objects (CBO), Response for Class (RFC), Lack of Cohesion between Methods (LCOM), and Weighted Method Complexity (WMC).

The same OO metrics were used as a system quality predictor ([35], [17], [34], and [9]). In most cases, quality was defined in terms of maintainability, modifiability, understand-ability, and testing and development effort.

In the context of analyzing the impact of design metrics on a system's modifiability, Deligiannis et al. [25] studied the impact of design heuristics, such as coupling and cohesiveness in design on the quality and maintainability of OO design. The study

showed that OO design heuristics can substantially predict system maintainability: A more maintainable design at an earlier stage can reduce maintenance effort at a later stage. It was also found that high coupling between classes is the most significant factor that reduces maintainability.

Throughout our investigation in the related research of modifiability estimation, we found four important object-oriented metrics used to predict software systems modifiability, which can be easily adapted to our domain and used with OOFeM. These metrics were also identified during our investigation into the answers for the GQM questions described in Section 6.5. Table 6-1 provides more information about these metrics and their impact on modifiability.

Table 6-1: Object Oriented Modifiability Metrics

Metric	Impact on Modifiability
Complexity	• Complexity of entities is a predictor of how much time and effort is required to develop and maintain them [11]. • More complex entities need more time and effort in development, maintenance, and modification [35].
Inheritance	• The deeper an entity is in the inheritance hierarchy, the greater the number of characteristics it is likely to inherit, making it more complex [21]. • If a modification is made to an entity, all of its descendents need to be tested to ensure the desired functionality is maintained [70].

Metric	Impact on Modifiability
Coupling	• The higher the number of links between entities, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult ([16] and [48]). • The higher the coupling, the more rigorous the testing needs to be once a modification occurs [50].
Cohesion	• Cohesiveness between components of an entity is desirable since it increases modifiability [15]. • If components are similar, then any modification to some of them can be replicated in others. This will reduce the effort of modification and testing ([48] and [50]). • Lack of cohesion implies that entities are assigned too many duties and should be split into two or more sub-entities because the class must provide logically related functionalities [56].

We will use an adaptation of these metrics in our approach for estimating the impact of features on system modifiability.

6.4 Step 1: Modeling System Features using OOFeM

The first step starts by modeling the features using the OOFeM discussed in Chapter 5 in order to extract the Feature-Oriented Modifiability Metrics (FOMM) (discussed in the next Section). This is because OOFeM is adapted from the object-oriented domain and considers external modifiability metrics, such as coupling and precedence, as well as internal modifiability metrics, such as complexity and cohesion.

Here, a variety of UML tools can be used to support this, such as ArgoUML [1] or IBM Rational Rose [2]. Models created with these tools enable the automatic generation of the FOMM metrics later (i.e. using the SDMetrics [4] tool)

6.5 Step 2: Extracting the Features Oriented Modifiability Metrics (FOMM)

In this section, we determine what we consider the most relevant OO metrics and adapt them to the release planning domain, in order to estimate a feature's impact on modifiability.

This thesis does not define new modifiability metrics, but rather adapts existing ones. In order to better identify which metrics to collect, we used the Goal Question Metric (GQM) approach ([10] and [92]). GQM helped us identify questions from our goal as drivers for finding the relevant metrics. The goal is to improve (purpose) the modifiability (issue) of the system (object) from the project manager's viewpoint.

The following questions were identified which characterize how the goal should be attained:

1. What is the current impact of a feature on system modifiability?
2. What is the current modifiability of the system?
3. How do features interact with other features? In other terms, what kind of dependencies exist between features? How do these dependencies impact system modifiability?
4. How can a feature's internal characteristics impact system modifiability?

Based on these questions, we use four OO design metrics adjusted to the feature domain as Feature-Oriented Modifiability Metrics (FOMM). The formulas for deriving these

metrics are adapted from([17], [21] ,[34], [54] and [56]) and are described in the following subsections.

6.5.1 Feature Complexity Metric (FCom)

FComp indicates how complex a feature is to modify in isolation. We use Bunge's definition of the complexity of things [99].

Definition 6.1: Let *Req(n) = {r(n,1), r(n,2), ..., r(n,TR(n))}* be the set of requirements in feature *f(n)*, effort*(r(n,i))* be the effort needed to implement requirement *r(n,i)* of feature *f(n)*. Then, feature complexity metric *FComp* is estimated as follows:

$$FComp(n) = \Sigma_{i=1..TR(n)} \text{ effort}(r(n,i)) \qquad \textbf{(6-1)}$$

6.5.2 Feature Inheritance Metric (FInh)

FInh is derived from the precedence relationship. The greater the number of descendants (or preceding features) the greater the effort needed to test the children when a modification happens to a feature. Also, the deeper the hierarchy tree, the greater the number of operationalisms that may be reused, making it more difficult to accommodate change requests ([9] and [21]).

Definition 6.2: Let *F = {f(1), f(2), ..., f(M)}* be the set of features. Then, the feature inheritance metric *FInh* is estimated as follows:

$$FInh(n) = \Sigma_{i=1..M,\ i \neq n} \ isDescendant(f(i),f(n)) \qquad \textbf{(6-2)}$$

Where

$$isDescendant(f(i), f(n)) = \begin{cases} 1 & if\ (f(n)\ precedes\ f(i)) \\ & or\ (f(i)\ precedes\ f(n)\ and \\ & f(i)\ is\ already\ implemented) \\ 0 & otherwise \end{cases} \qquad \textbf{(6-3)}$$

6.5.3 Feature Coupling Metric (FCoup)

FCoup is derived from the coupling dependency between features. The higher the coupling, the higher the sensitivity to changes in other features and therefore accommodating change requests becomes more difficult. Also, the higher the coupling, the more rigorous the testing needs to be, which increases the total modification effort ([16], [34], and [50]). For estimating *FCoup*, we use associations between features in OOFeM as an indication of coupling dependencies as follows:

Definition 6.3: Let $F = \{f(1), f(2), ..., f(M)\}$ be the set of features. Then, the feature coupling metric *FCoup* is estimated as follows:

$$FCoup(n) = \Sigma_{i=1..M}\ isCoupled(f(i),f(n)) \qquad \textbf{(6-4)}$$

Where

$$isCoupled(f(i), f(n)) = \begin{cases} 1 & if\ (f(i)\ is\ coupled\ to\ f(n)) \\ & of\ (f(n)\ is\ coupled\ to\ f(i)\ and \\ & f(i)\ is\ alrady\ implmented) \\ 0 & otherwise \end{cases} \qquad \textbf{(6-5)}$$

6.5.4 Feature Cohesion Metric (FCoh)

FCoh is derived from requirements and operationalisms in features. Since operationalisms were used to satisfy requirements and describe them operationally in OOFeM, *FCoh* indicates the degree to which the operationalisms address the same requirement. The more operationalisms address the same requirement, the more cohesive the feature is, which means less effort is needed to modify its requirements ([15], [21], and [50]). *FCoh* is estimated as follows:

Definition 6.4: Let *Req(n)* = {*r(n,1), r(n,2), ..., r(n,TR(n))*} be the set of requirements in feature *f(n)* where *TR(n)* is the total number of requirements in *f(n)*, *Oper(n)* = {*o(n,1), o(n,2), ..., o(n,TO(n))*} where *TO(n)* is the total number of operationalisms in *f(n)*. Then, the feature cohesion metric *FCoh* is estimated as follows:

$$FCoh(n) = \frac{\sum_{i=1}^{TO(n)} \sum_{j=1}^{TR(n)} isAddr\ (o(n,i), r(n,\ j))}{TO\ (n)\ x\ TR\ (n)} \qquad \textbf{(6-6)}$$

Where

$$isAddr(o(n,i),r(n,j)) = \begin{cases} 1 & \text{if } o(n,i) \text{ addresses } r(n,\ j) \\ 0 & \text{otherwise} \end{cases} \qquad \textbf{(6-7)}$$

Once the OOFeM model has been created using UML design tools, most of these metrics can be automatically generated with the help of the OO metrics tools, such as SDMetrics [4]

6.6 Step 3: Modifiability Indicator (IMod) Estimation

We are concerned with how the implementation of one feature impacts other features as well as the possibility of accommodating change requests in future. Thus, the aforementioned four metrics which cover internal and external feature modifiability characteristics are combined in order to estimate a feature's impact on modifiability. To ensure comparativeness between the different values, we normalize all metrics within the range [0, 1]. The Euclidean distance to the ideal point, *Dist(n)* is calculated [95] with respect to metrics relevant for determining a feature's impact on modifiability. The ideal point is given by a feature with the lowest coupling, lowest inheritance, lowest

complexity, and highest cohesion. The concept of ideal point minimization is discussed in more detail in Section 7.3.1.

Features with a low *Dist(n)* are considered more attractive in terms of their predicted impact on modifiability. Thus, the modifiability indicator (*IMod(n)*) is defined as follows:

Definition 6.5: Let *normFComp(n)* be the normalized value of *FComp(n)* within the range [0,1], *normFInh(n)* be the normalized value of *FInh(n)*, *normFCoup(n)* is the normalized value of *FCoup(n)*, *normFCoh(n)* is the normalized value of *FCoh(n)*. Then, the modifiability indicator *IMod(n)* of feature *f(n)* is estimated as follows:

$$IMod(n) = 1 - normDist(n) \quad \textbf{(6-8)}$$

Where normDist(n) is the normalized value of Dist(n) and estimated as follows:

$$Dist\ (n) = \sqrt{a^2 + b^2 + c^2 + d^2} \quad \textbf{(6-9)}$$

Therein,

a = normFCom(n);

b = normFInh(n);

c = normFCoup(n);

d = (1 - *normFCoh(n)*)

Chapter Seven: Reactive Decision Support for Handling Change Requests (RESUP)

In this chapter, we present the second part of our approach, which is concerned with reactive decision support for handling change requests (RESUP). RESUP handles change requests after their arrival by relying on a re-planning method called H2W. We start by providing an overview of H2W, including the research questions addressed. After that, we discuss in more detail the technical aspects of each question.

7.1 Overview of H2W

H2W is a re-planning method that provides decision support for software release planning after the arrival of change requests. It starts by specifying the re-planning condition. Once this condition is satisfied, the re-planning is triggered. Consequently, the method generates a new release plan. Finally, the method tries to find a compromise between the added new value and release stability.

H2W answers the questions of how, when, and what to re-plan for an existing software release. For HOW, a greedy heuristic based on prioritization of candidate features, based on their distance to the ideal point, finds the best compromise between features' value, effort and risk. An approximation for the likelihood of change in software releases as well as a value-based re-planning approach is proposed for the WHEN question. For WHAT, a trade-off analysis between the degree of change related to the originally announced release plan and the value improvement achieved by replacing existing features with more attractive ones is suggested.

H2W addresses the above questions in an iterative manner and considers the most recently created plan as a baseline for the next iteration. At each re-planning iteration,

H2W either provides a new improved release plan or states that an improvement does not exist. The workflow of the overall method is shown in Figure 7-1.

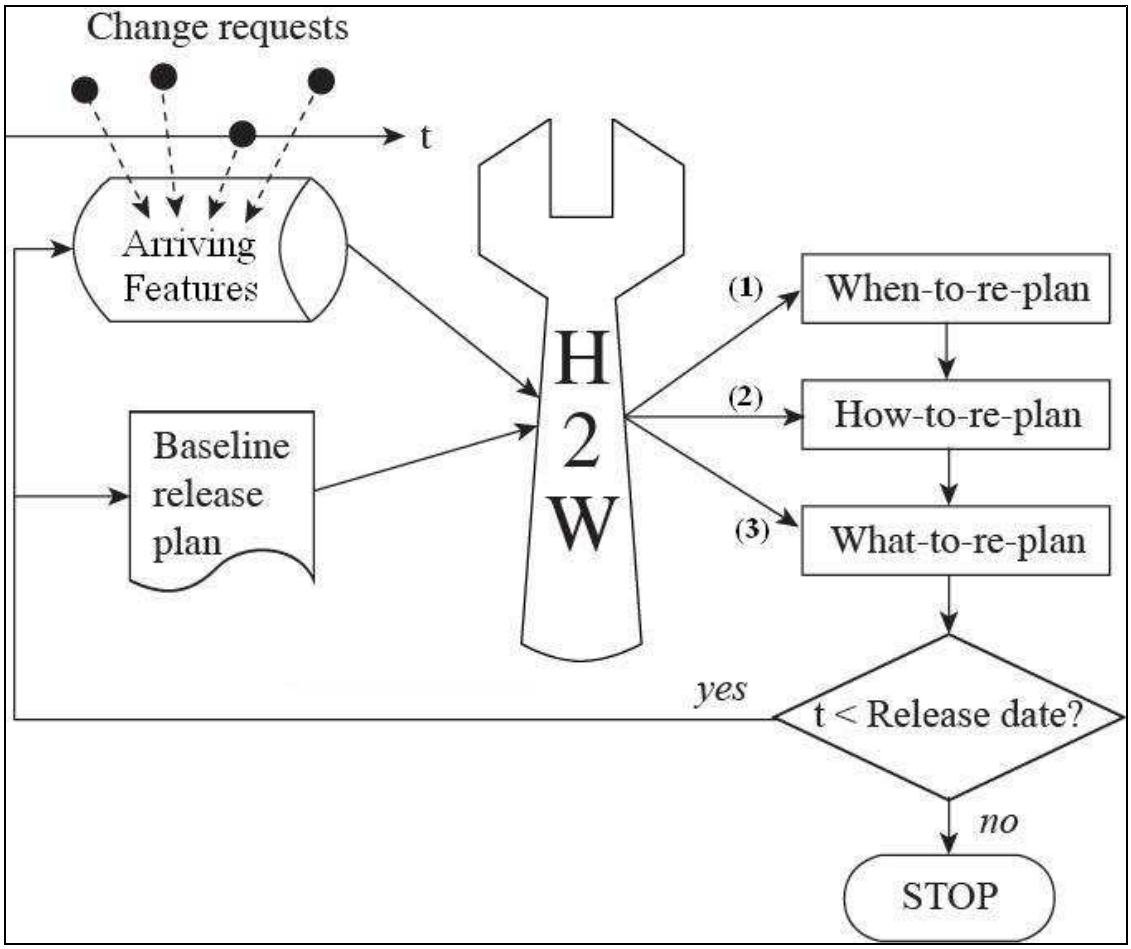

Figure 7-1: Workflow of the H2W method

The change requests arrival with their cumulative impact on the release value is illustrated in Figure 7-2. Each step of the curve corresponds to the arrival of a new change request which adds value to the release. However, H2W only considers a portion of these new features to incorporate into the existing plan. In the illustration, we have assumed two re-planning steps. Their timing within the release interval [T1,T2] is highlighted by the two black arrows at the x-axis.

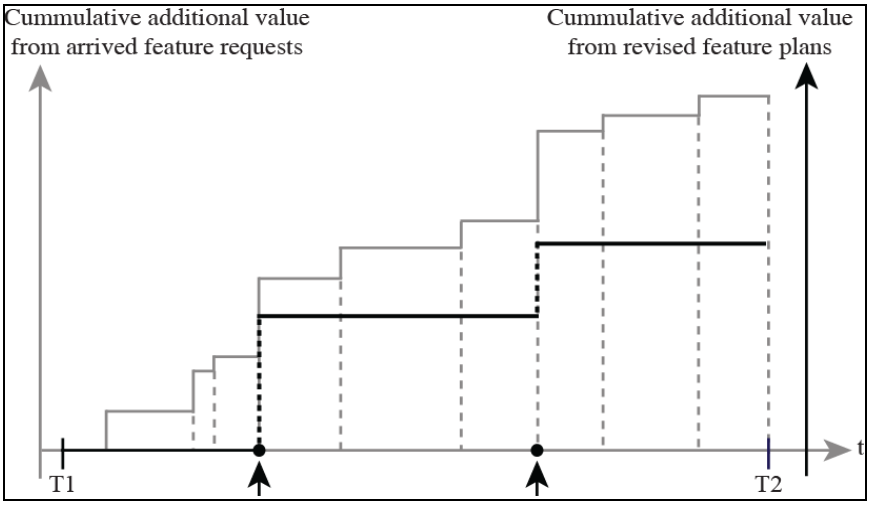

Figure 7-2: Impact of arriving features and change requests on the release value

7.2 WHEN to Re-plan?

Three issues need to be addressed for the question of when to re-plan:

1. The attractiveness (in terms of value) of new features: Some of the new features and change requests may be of substantial value to the customer and the organization.
2. The openness for change: As we get closer to the release date (i.e. t → T2), changes become more difficult to accommodate within the limited time left. Also, studies show that the closer changes take place to the release date, the greater the density and severity of defects ([40] and [57]).
3. The cumulative amount of change: This is for situations where re-planning is triggered more than one time during the same release. In this case, the degree of change in the baseline release plan will be increased. Our assumption is that changing the baseline release plan often is not recommended.

For the first issue, different metrics could be considered to decide how attractive new features are. For this purpose, we consider two metrics:

1. Accumulated value of new features: Since each feature has its own value, we consider a value based trigger for re-planning. For example, it may be decided to trigger re-planning once the accumulated value of new features reaches 25% of the baseline plan value.
2. Number of new features: Since features are accumulated before being considered for re-planning, we use a cardinality based trigger (Card) for re-planning. For example, it may be decided to trigger re-planning once the cardinality of new features reaches 25% of the baseline plan cardinality.

The actual value for each metric is project and context specific and is defined by the product manager, taking into account his or her former experience, as well as the current business and market conditions.

The "openness to change" decreases over time since there is less willingness to re-plan. The actual decrease function varies from project to project, depending on the circumstances of the organization and their customers. At the beginning of the release period (i.e. t = T1), changes are easier to accommodate than towards the end (i.e. t = T2). We assume that the decrease follows a linear function *h(t)* with a negative slope [82]. *h(t)* is defined for the planning interval [T1,T2] as:

$$h(t) = - \lambda \, (t - T1) \, / \, (T2 - T1) + \lambda \qquad \textbf{(7-1)}$$

Where:

- λ is a factor [0-1] used to adjust the intial level of openess to change to the project context and organizational settings.

At any point in time, the function value *h(t)* gives the percentage of change in the baseline release plan features F_b, for which there is enough reason to re-plan and replace them by newly arrived ones. The more time passes, the less acceptable is the re-planning. Figure 7-3 shows *h(t)* for T1 = 0, T2 = 100, $\lambda = 0.5$.

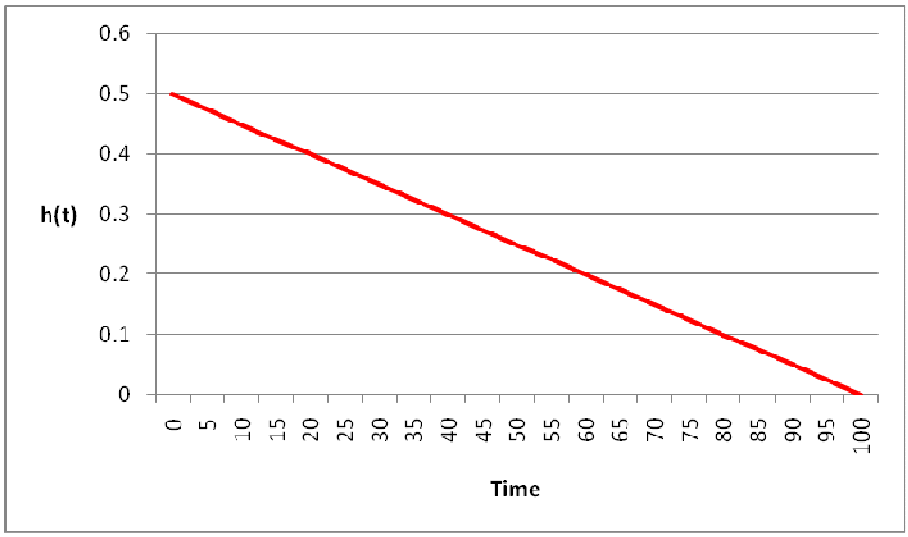

Figure 7-3: Openness to change function h(t)

The re-planning condition:

The first re-planning is triggered at t, where t is the earliest point in time when one of the following conditions is satisfied:

1. Value based trigger:

$$[Value(CR(t))/Value(F_b)] \geq h(t) \quad \textbf{(7-2)}$$

2. Cardinality based trigger:

$$[Card(CR(t))/Card(F_b)] \geq h(t) \quad \textbf{(7-3)}$$

Where *CR(t)* is the set of all change requests and new features arrive until *t*. F_b is the set of features in the baseline release plan.

The key steps of the method for the two types of triggers are described in more detail in pseudo-code representation in Figure 7-4. First of all the value of h(t) is evaluated at the time when the re-planning condition is tested. Then, based on the type of trigger used for re-planning, either one of the following is evaluated:

- If the cardinality based trigger is used: calculate the ratio between the number of new features and the number of features in the baseline release plan.
- If the value based trigger is used: calculate the ratio between the added value of new features and the baseline release plan value.

If any of the values evaluated above exceeds or equals the value of h(t), then there is enough reason to re-plan. Hence, re-planning is triggered.

```
Initialize  BaselineValue to 0 // The baseline release plan total value
Initialize  TotalNewValue to 0 // Accumulative added values of new features and
change requests.
Initialize  DoReplanning to false // Determine when we can do re-planning

Set         h(t) = - λ ( t – T1) / (T2 – T1) + λ
If          (UsedTrigger = CardinalityBasedTrigger)
Then        If          ( ((Nt-M)/(P)) >= h(t) )
            Then        Set DoReplanning = true   // start re-planning
            Endif
ElseIf      (UsedTrigger = ValueBasedTrigger)
Then        For         n = 1 to P  // for each feature in the baseline plan
                        BaselineValue = BaselineValue + value(n)
            Endfor
            For         n = M+1 to Nt  // for each new feature & change request
                        TotalNewValue = TotalNewValue + value(n)
            Endfor
            If          ( (TotalNewValue/ BaselineValue) >= h(t) )
            Then        Set DoReplanning = true   // start re-planning
            Endif
Endif
```

Figure 7-4: Pseudo code for triggering re-planning

7.3 HOW to Re-plan?

In this step, the actual re-planning is done. For this purpose, candidate features are ranked based on their distance to the ideal point. Greedy optimization is applied in order to assign the most attractive features to the remaining capacities. The operational plan [52] is evaluated at the time of re-planning to keep track of features' status (i.e. implemented, not implemented, being implemented) as well as to estimate the remaining capacity for the rest of the release period.

We will discuss these concepts in detail in the next sub-sections.

7.3.1 Distance to Ideal Point Minimization

We use the concept of minimizing the distance to the ideal point known from multi-criteria optimization [95] for determining the most attractive features. The ideal solution is an artificial solution defined in the criterion space, the elements of which determine the optimum of a criterion's value. As this artificial solution cannot be achieved in most cases, the goal is to come as close as possible to the ideal solution. The distance can be measured, for example, using the Euclidean distance. The solution being closest to the ideal is considered to be the best compromise and the most promising one to pursue.

The concept is illustrated in Figure 7-5. To keep it simple, only two criteria were selected (i.e. effort and value). We calculated the Euclidean distance to the ideal point. The ideal point is given by the coordinates of a feature with (value, effort) = (9,1). For features with associated points a, b, c, d, e, and f, the ranking of features would be: b » d » e » f » c » a, where » stands for "closer to ideal point".

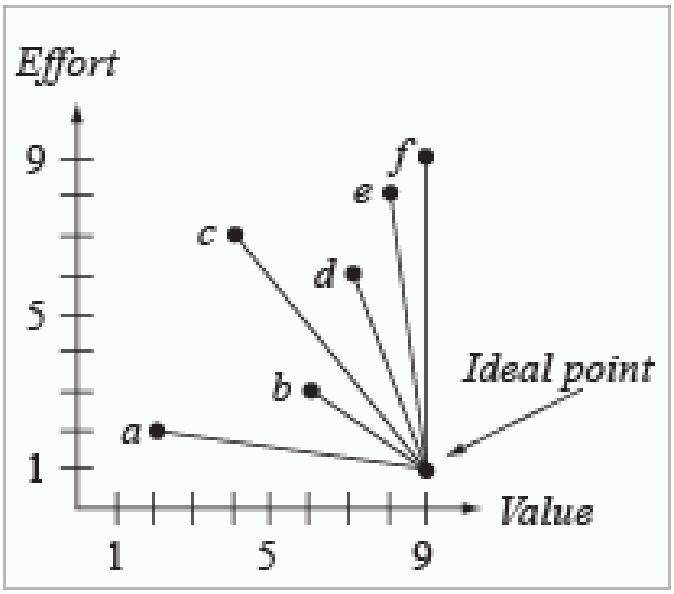

Figure 7-5: Minimization of distance to ideal point

In our case, we try to minimize the distance to the ideal point in a four-dimensional space defined by the features' attributes: value, effort, risk, and impact on modifiability. According to the measurement theory [28], the differences and ratios between values in the ordinal scale are not considered at all, which makes it very difficult to provide an objective function to measure the distance. Hence, for simplicity reasons, we follow the common practice in software engineering [82], by re-interpreting the ordinal nine points scale into a ratio scale in the range of the used nine point scale, and measure the distance. The feature f(n) that minimizes the distance to the ideal case of having the lowest effort, lowest risk, lowest impact on modifiability, and highest value is considered to be of highest priority (i.e. *priority(n)* is maximized) and the best candidate for inclusion into the next release plan.

7.3.2 Greedy Method for Release Re-planning

Greedy algorithms are widely applied to iterative solution techniques aimed at finding a good global solution by determining the local optimum [22]. The key principle of greedy release planning is to sort features top down according to their overall priority. In our approach, the higher the priority, the lower the distance to the ideal point.

The optimization process starts by assigning the feature with the highest overall priority to the current release. This process is continued with the next best feature. Each high-priority feature is assigned to the current release as long as the available resource capacities are not exceeded. If the addition of any of the new features would violate any of the capacity constraints, then the feature is dismissed and the next best feature is considered for inclusion. The process terminates when no further feature can be added to the existing set within the remaining capacity.

Since feature dependencies (i.e. coupling and precedence) affect the assignment of features to different releases, these dependencies are handled during greedy re-planning as follows:

- If feature f(i) is coupled to f(j), then:
    - If priority(i) > priority (j), then both are combined into one big feature f(k) with a total effort and average priority of both features as well. In cases where multiple features are coupled to one features, the same process is applied just for the highest priority feature, while subsequent features are treated individually.
    - If priority(i) < priority (j), nothing is done since the feature with higher priority is assumed to be implemented earlier.

- If feature f(i) precedes feature f(j), then:
    - If priority (i) > priority (j), then nothing is done since the feature with higher priority is assumed to be released earlier.
    - If priority (i) < priority (j), then both are combined into one big feature f(k) as done with coupling dependencies.

The key steps of the method are described in more detail in a pseudo-code representation in Figure 7-6. As presented in the figure, the remaining capacity *RemCAP* is estimated at the beginning of re-planning process, from developers available during the remaining time for the release. After that, distance to the ideal point *Dis(n)* is calculated for all features included in re-planning. Features and their distances are added to a temporary list, called *Features*. This list is sorted ascending according to a feature's distance from the ideal point.

Before applying greedy heuristics to update the current release plan, coupling and precedence dependencies between features are considered as described above. This may results in affecting the priorities of features or combining multiple features into a one feature that compromise their effort and priority. Finally greedy algorithm is applied adding features with high priority (i.e. top features in the sorted *Features* list) to another list, called *CandidateFeatures*, which represents candidate features to be included in the rest of the release. The re-planning stops when no further feature can fit in the remaining capacity *RemCap* of the release.

```
Initialize CandidateFeatures to empty // List containing the best candidate features
Initialize AvgNDev // Average no. of developers available over planning period
Initialize Features to empty // Temporary list containing the features.

Set     RemCAP = (T2 - t) * AvgNDev // estimating the remaining capacity of the release
For     n = N+1 to N_t // prioritize features according to their distances from the ideal point
        Normalize effort(n) in a scale of {1,..,9} and store it in normEff(n)
        If      ((f(n) is not implemented ) OR (f(n) is change request))
        Then    Set Dis(n) = Sqrt((value(n)-9)^2+(normEff(n)-1)^2+(risk(n)-1)^2+(IMod(n)-1)^2)
                Features.add(f(n))
        EndIf
Endfor
Features.sortBy(Dis) // sort features according to their distances (priorities)
While   (RemCAP > 0) // greedy algorithm
        Set     featureFound = false
        For     n = 1 to  Features.Size()
                // Handle the coupling and precedence dependencies between features
                If      (Features[n].isCoupledTo(Features[i]) AND i > n) OR
                        (Features[i].Precedes(Features[n]) AND i > n)
                Then    Set Features[n]= Features[n] UNION Features[i]
                        Set Features[n].Dist= (Dist(Features[n]) + Dist(Features[i])) / 2
                        Set Features[n].Effort = (Effort(Features[n]) + Effort(Features[i]))
                        Features.remove(f(i))
                        Features.sortBy(Dis) // sort features again
                        // Stop the for loop and enforce it to rework after the new adjustment
                        Break
                Endif
                // Verify the resource constraints and add the feature to the next release
                If      (effort(Features[n]) < RemCAP )
                Then    CandidateFeatures.add(f(n))
                        RemCAP = RemCAP – effort(n)
                        Features.remove(f(n))
                        Set   featureFound  = true
                        Break  // stop the closest for loop
                Endif
        Endfor
        If      (featureFound = false)
        Then    Break // stop the while loop
        Endif
Endwhile
```

Figure 7-6: Pseudo code for the re-planning process

7.3.3 Evaluation of Operational Release Plan

After producing the baseline release plan (i.e. strategic plan), the product manager tries to implement it during the release period. Operational planning [64] describes a detailed operational procedure for implementing the announced plan. It defines what the project manager will do to implement the goals, what tasks are needed, what kind of dependencies exist between tasks, what capacities are needed and how resources will be allocated ([42], [64], and [52]).

An example of a simple operational plan is provided in Figure 7-7. As shown in the figure, we consider three types of resources: design, development, and testing resources. For this example, we assume the features are implemented in order of priority (i.e. distance to ideal point).

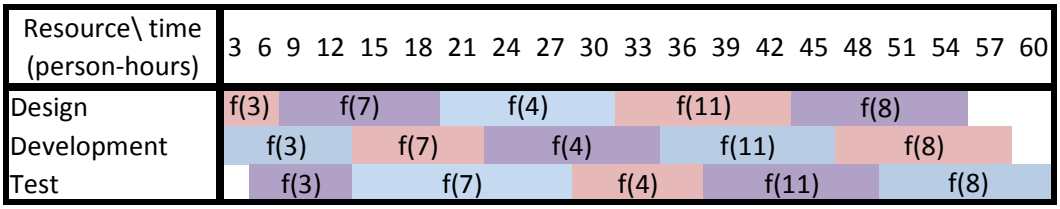

Figure 7-7: Operation plan example

The problem becomes visible at the time of re-planning as some features will be implemented, others will be under implementation, while others will be scheduled for implementation later in the release. The capacity of the remaining resources for the rest of the release period needs to be estimated. In addition, decisions need to be taken regarding features under development. Although our research is concerned with strategic planning for software releases, a simple form of operational planning is needed to keep track of the status of features during the release.

Our approach suggests doing the following at the time of re-planning:

1. For features under development, we calculate the percentage of completion which can be obtained by dividing resources consumed during the development of the feature until re-planning time by the estimated total resources needed for implementing the feature. If a feature is n % implemented, we keep it. Otherwise, we consider it for re-planning. The actual value of n is context specific and is defined by the project manager taking into account the value of the feature compared to the value of new change requests.
2. To estimate the remaining capacity for the rest of the release, we subtract the resources consumed during the development of already implemented features as well as features under development from the total release capacity CAP.

For example, in Figure 7-7, if re-planning is triggered at time t= 36, then features f(3) and f(7) are already implemented, f(4) and f(11) are being implemented, while f(8) is not yet implemented. Feature f(8) is considered for re-planning while nothing can be done for features f(3) and f(7). For f(4) and f(11), the percentage of completion is:

f(4) = 12(design) + 12(dev) + 6(testing) / 12(design) + 12(dev) + 9(testing) = 91%

f(11) = 3(design) + 0(dev) + 0(testing) / 12(design) + 12(dev) + 12(testing) = 8%

The project manager defined n = 75%. So, f(11) is considered for re-planning while f(4) is not. For the remaining release capacity:

Remaining design capacity = 60 - 30 = 30 person-hours.

Remaining development capacity = 60 - 33 = 27 person-hours.

Remaining testing capacity = 60 - 36 = 24 person-hours.

7.4 WHAT to Re-plan?

The purpose of release re-planning is to accommodate change requests in order to produce up-to-date release plans. However, modifying the baseline release plan frequently is not acceptable. There is a trade-off between increasing release value by accommodating more change requests on the one hand, and the need to keep the baseline release plan as stable as possible. Compromising between these two competing objectives enables us to update release plans while maintaining their stability requirement.

Definition 7.1: Let CR(t) be the set of change requests until re-planning time t, F_r be the set of rejected features during baseline release planning, F_b be the set of features in the baseline release plan. Then, there exists a set of P features R_f = {fr(1),..., fr(P)} to be removed from the baseline release plan, and a set of new Q features N_f = {nf(1),..., nf(Q)} to replace the removed features in R_f as a result of re-planning such that:

$$R_f \subseteq F_b \quad \textbf{(7-4)}$$

$$N_f \subseteq (CR(t) \cup F_r) \quad \textbf{(7-5)}$$

The priority and effort constraints can be described as follows:

$$\Sigma_{n=1..Q}\ \text{effort}(nf(n)) \leq \Sigma_{n=1..P}\ \text{effort}(rf(n)) \quad (\textbf{7-6})$$

$$\Sigma_{n=1..Q}\ \text{priority}(nf(n)) \geq \Sigma_{n=1..P}\ \text{priority}(rf(n)) \quad (\textbf{7-7})$$

Definition 7.2: Let R_f be the set P features to be removed from the baseline release plan, N_f be the set of new Q features to replace the removed features in R_f as a result of re-planning. Then the added value to the baseline release plan as a result of re-planning can be estimated by:

$$\text{AddedReleaseValue} = \Sigma_{n=1..Q}\ \text{value}(nf(n)) - \Sigma_{n=1..P}\ \text{value}(rf(n)) \quad (\textbf{7-8})$$

Definition 7.3: Let R_f be the set P features to be removed from the baseline release plan, N_f be the set of new Q features to replace the removed features in R_f as a result of re-planning. Then the set of all replacements can be described by $\Phi = \{\Phi(1), ..., \Phi(P)\}$ where $\Phi(i)$ represents a relation between one feature $R_f'(i)$ from R_f and a set $N_f'(i)$ of k feature from N_f such that:

$$\text{effort}(R_f') \geq \Sigma_{n=1..K} \text{ effort}(N_f'(n))$$
$$R_f'(i) \in \Phi(i), N_f'(i) \in \Phi(i), K \geq 1 \quad (\textbf{7-9})$$

The added value of this replacement can be estimated by:

$$\text{AddedValue} = \Sigma_{n=1..K} \text{ value}(N_f'(n)) - \text{value}(R_f'(n)) \quad (\textbf{7-10})$$

The question "What to re-plan?" determines what features shall be kept in the re-planned release, as a result of feature replacements, which do not violate the stability constraint of the release. This is achieved by performing a trade-off analysis between the relative increases in the baseline release value to the relative decrease in release stability.

The trade-off is conducted as follows:

1. Relations in Φ are sorted in descending order according to their added values.
2. Replacements are done in the same order of sorted Φ (i.e. top element in Φ is replaced first while the bottom element is replaced last).
3. Relative increase in value is plotted against the relative decrease in release plan stability as a result of feature replacements.
4. This is done for different numbers of feature replacements.
5. The point of intersection between these curves determines the best number of features to be replaced from the baseline plan.

The key steps of the method are described in more detail in a pseudo-code representation in Figure 7-8. As show in the figure, features in R_f and N_f are sorted at the beginning in descending order according to their values. After that, the best arrangement of feature replacements (i.e. replacements that provide the highest increase in value) is determined by replacing features from the top of R_f (i.e. highest value) with features from the bottom of N_f (i.e. lowest value). This is done for different cardinalities of feature replacements. For every cardinality of replacement, the relative added value and the relative decrease in stability is calculated. The relative values are then plotted as two curves in the trade-off figure. The point of intersection between the two curves represents the best compromise between adding new features with high value and modifying the baseline release plan.

Precondition 1: Set of features R_f = {*fr(1) …fr(P)*} to be removed from baseline plan as a result of Step 2.
Precondition 1: Set of new features N_f = {*nf(1) …nf(Q)*} to replace features in R_f as a result of Step 2.

Initialize *cumIncreaseInValueList* to empty
Initialize *cumDecreaseInStabilityList* to empty

R_f.sortInDecreasingOrderBy(Value)
N_f.sortInDecreasingOrderBy(Value)
For *n = 1 to P // for each number of replacement from the baseline plan*
➔ Determine the best n replacement from the top of R_f (called R_f') with the bottom of N_f (called N_f') such that:
$\Sigma_{i=1..q'}$ effort(N_f'(i)) ≤ $\Sigma_{i=1..p'}$ effort(R_f'(i))
cumIncreaseInValueList [*n*] = $\Sigma_{i=1..q'}$ value(N_f'(i)) - $\Sigma_{i=1..p'}$ value(R_f'(i))
cumDecreaseInStabilityList [*n*] = *1 – n/P*
Endfor

Determine p* to be the index such that the two points (p*, *cumIncreaseInValueList*[p*]) and (p*, cumDecreaseInStabilityList[p*]) are closest to each other.// This represents the best compromise.

Figure 7-8: Pseudo code for value-stability trade-off

Chapter Eight: Case Study

8.1 Introduction

In this chapter, we present our case study, which was conducted in Semantic Designs Inc., in order to evaluate the applicability of our Sup-HCR approach. Semantic Design is a software company mainly working in the analysis, modification, translation and generation of large scale software systems. Our approach was applied during the process of handling change requests in the planning and re-planning for the next release for a commercial tool, called CSharp Test Coverage. A tool which permits the creation and display of detailed information about which parts of a CSharp application program have been executed.

The case study is further divided into two steps: The first step is concerned with evaluating the applicability of our proactive decision support method (PROSUP), by considering the modifiability in a real world release planning scenario. The output of the first part, besides simulated data representing new change requests are used in the second part in order to evaluate the applicability of our reactive decision support method (RESUP).

8.2 Case Study Design and Planning

Following the instructions and guidelines provided by Runeson et al. for conducting and reporting case study research in software engineering [85], our case study design and planning can be summarized as follows:

- Object: what is studied?
- Objective: what to achieve?
- Research questions: What to know?

- Data collection methods: how to collect data?
- Selection strategy: where to seek data?
- Stakeholders: who are involved?
- Tool support: what automation was provided?

8.2.1 Case Study Object

This case study is concerned with the process of handling change requests in planning and re-planning for the next release, for the CSharp Test Coverage tool, in Semantic Design Inc. We were interested in studying the outcomes of the release planning with and without applying our approach, as well as studying the effectiveness of our release re-planning method. A screenshot of the developed tool is provided in Figure 8-1. As shown in the figure, the tool consists of three frames. The upper frame is used to manage the Test Coverage (TCV) computations and arithmetic, while the lower right frame shows the results of these computations superimposed on the source code. The lower left frame allows the user to navigate through covered and uncovered blocks of source code.

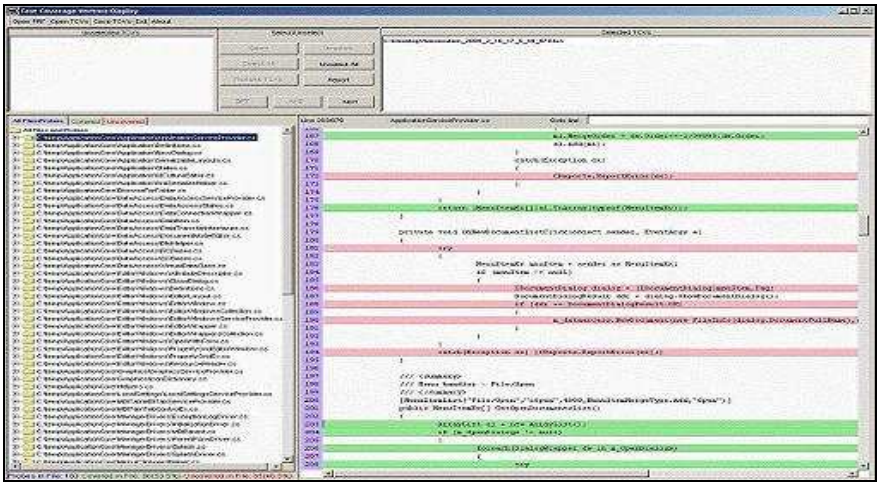

Figure 8-1: Screenshot of the developed C# test coverage tool

8.2.2 Case Study Objective

This case study was mainly conducted to investigate the applicability of our Sup-HCR approach for handling change requests. Hence, the objective was exploratory [80]. We wanted to apply this approach in a real world release planning and re-planning situation and observe the outcomes and how applicable are our suggestions?

8.2.3 Research Questions

The research questions state the open issues, which need to be addressed in order to fulfill the objective of the study [85]. The following questions were identified and evolved over the case study:

Question 1: *How close our approximation of features in terms of their composition and interaction to that in reality*?

Question 2: *How applicable is producing the OOFeM from real world software systems' documents and specifications? And how applicable is calculating the modifiability metrics (FOMM) out of the OOFeM models produced (if any)?*

Question 3: *In the absence of modifiability concern, does conducting systematic release planning have any advantages over not conducting it (ad-hoc release planning)?*

Question 4: *In the case of systematic release planning, does addressing modifiability concern have any advantages over not addressing it?*

Question 5: *How applicable is our reactive decision support method for release re-planning in reality?*

8.2.4 Data Collection and Selection Methods

The data used in this case study was primarily collected using:

1. Interviews: the decisions makers were interviewed to discuss their concerns with the process of handling change requests when planning and re-planning for the next release.
2. Interaction with the tool: The first set of features was elicited by interacting with the tool and experiencing its functionalities.
3. Archival data analysis: Some documents were analyzed in order to obtain the full list of features as well as validating the initial set produced while interacting with the tool as follows:
    a. C# test coverage tool documentation.
    b. C# test coverage tool requirements document.
    c. C# test coverage tool functionalities.

8.2.5 Stakeholders

In this case study, two stakeholders were involved in four roles during the release planning process, as described in Table 8-1.

Table 8-1: Stakeholders ivolved in the case study

ID	Assigned roles
S1	- Chief Executer Officer (CEO) - Project Manager (PM)
S2	- Requirements Engineer (RE) - Software Architect (SA)

8.2.6 Tool Support

The following tools were used to help conducting this case study:

1. ArgoUML: used to draw the features diagram for OOFeM.
2. SDMetric: used to help in calculating the FOMM metrics after exporting the OOFeM structure generated in (1).

8.3 Case Study Protocol

Case study protocol is considered as a reference for researcher on how to conduct the case study. It describes the design and steps need to be conducted ([75], [85]).

Figure 8-2 provides an overview of the case study protocol. It contains the following steps:

1. Step 1- Interviewing decision makers: the CEO and project manger were interviewed in order to explain the decision support method Sup-HCR, and to agree on subsequent steps needed to conduct the case study.
2. Step 2 – Eliciting system's features: three rounds of elicitation were conducted, in order to obtain the system' features, their requirements and operationalisms. These round were validated by the requirement engineer RA and the software architect SA as follows:
    a. Round 1: eliciting the initial set of features after reading the tool documentation and interacting with the tool.
    b. Round 2: eliciting the rest of features and their requirements after addressing the raised comments and reading the tool functionality document.

   c. Round 3: eliciting features operationalisms from tool requirements documents.

3. Step 3 - Effort estimation: meetings were held between the project manager, the requirements engineer and the software architect, in order to estimate the required cost in USD to implement the elicited features.
4. Step 4 - Features prioritization: all stakeholders were involved in prioritizing features according to their value, risk and complexity. These prioritizations are necessary for performing release planning and re-planning.
5. Step 5 – OOFeM construction: OOFeM was constructed using the previously elicited features' information, in order to be used in modifiability estimation.
6. Step 6 – Features' impact on system modifiability was estimated using the FOMM metrics extracted from the OOFeM which was generated in the previous step.
7. Step 7 – Baseline release planning for the next release is performed with and without considering a feature's impact on system modifiability.
8. Step 8 – Change requests arrival was simulated and re-planning was triggered at certain point in time.
9. Step 9 – Release re-planning was performed using the H2W method.

After conducting these steps, the results were analysed and conclusions were drawn regarding the applicability and usefulness of our Sup-HCR approach.

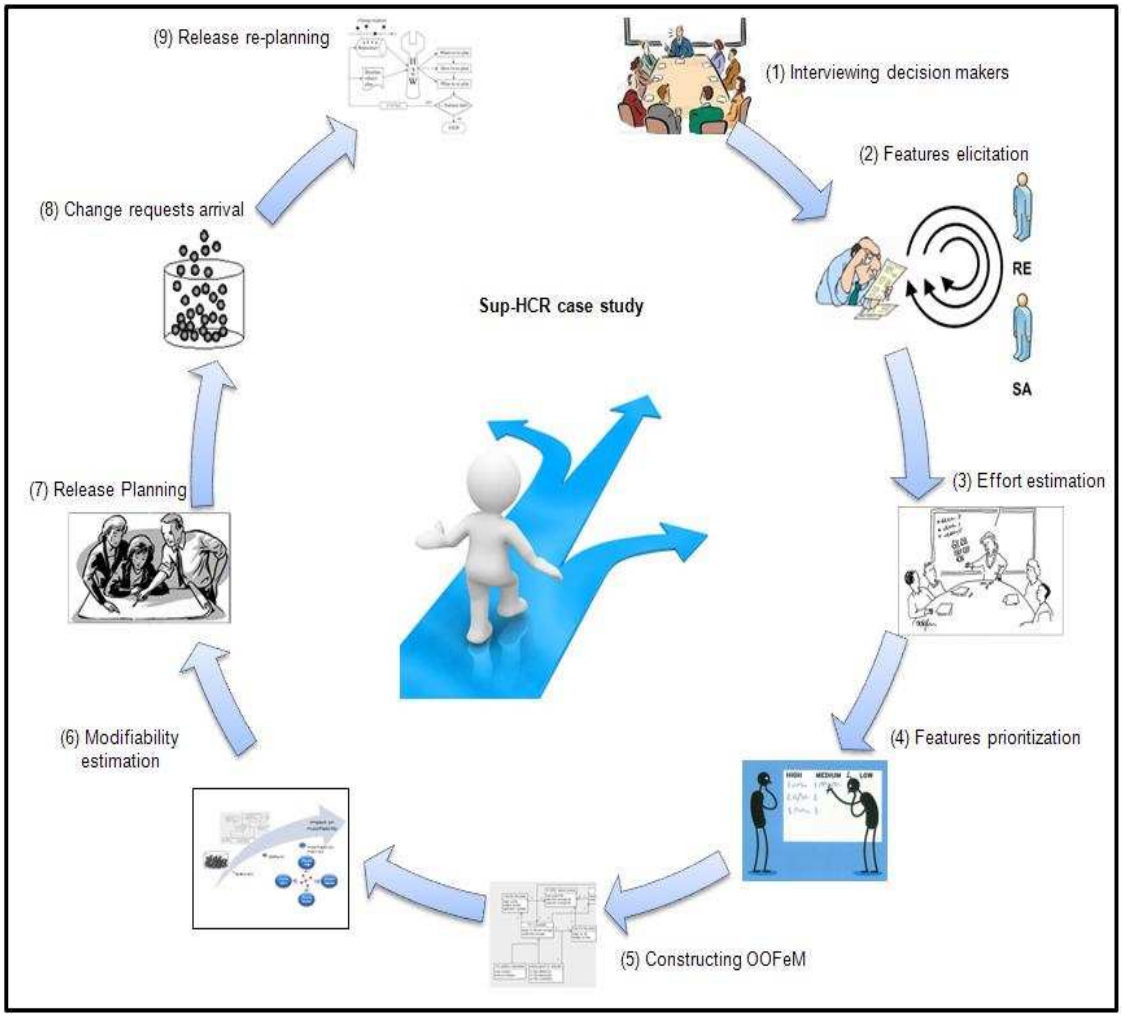

Figure 8-2: Overview of the case study protocol

8.4 Data Collection

Several sources were used to collect data used in this case study, in order to minimize the impact of the misleading interpretation that may be caused by using single data source [85]. We have used a combination of the first and the third level of data collection techniques described by Lethbridge et al [53] as follows:

- Level 1- Direct methods: data was collected when the researcher is in direct contact with the subject, such as during interviews and surveys.

- Level 3 – Analysing archival data: data was collected when researcher analyzed already available artefacts of the system, such as requirements documents and bug reports.

The following sub-sections discuss these data collection techniques in more detail.

8.4.1 Interviews

In our case study, part of the data was collected during the interview with both the CEO and the project manager (Level 1). The interview was semi structured [80] as the interview questions were not completely planned in advance. In fact, several topics were planned to be discussed in details during the interview such as planning and re-planning criteria, type of resources considered, applicability of the approach and agreement about subsequent steps needed in order to conduct the case study. Funnel model [85] was used to structure this interview, in which we started by asking open questions and then moved gradually towards more specific ones.

Unfortunately, since the interview was during a scientific conference, it wasn't recorded, but rather memos were taken for questions and their answers, as well as further concerns of decision makers.

Following the description of semi-structured interviews by Hove et al. [36], our interview can be summarized as follows:

1. Phase 1: An overview of our approach was presented to the decision makers (CEO and project manager), in 15 minutes. This helped to determine possible data sources at a later stage, as well as the specific tool (object) to be studied.
2. Phase 2: Objective of the case study was presented to the decision makers.

3. Phase 3: A set of introductory questions were asked about the background of the subject, which is mainly related to the release planning and re-planning processes. The following questions were discussed:
    a. *Q1: How release planning and re-planning are usually done in your company?*
    b. *Q2: Do you consider features' modification effort when assigning features to different releases?*
    c. *Q3: What are the criteria you usually use when performing release planning and re-planning?*
4. Phase 4: Main interview questions were asked, which mainly aimed at investigating the applicability of the approach. The following questions were discussed in details:
    a. *Q4: To what extent do you think considering a feature as a collection of requirements & operationalisms is applicable in real world?*
    b. *Q5: How can a feature representation in OOFeM be constructed in the real world? In other words what type of documents can be used to construct this model?*
    c. *Q6: Do you think considering a feature's impact on system modifiability early pays off at a later stage?*
    d. *Q7: What are the key issues to consider in software release re-planning?*
5. Phase 5: Case study protocol, which summarizes the steps needed in order to conduct the case study, which were agreed upon between the researcher and the decision makers. This protocol is further described in Section 8.3.

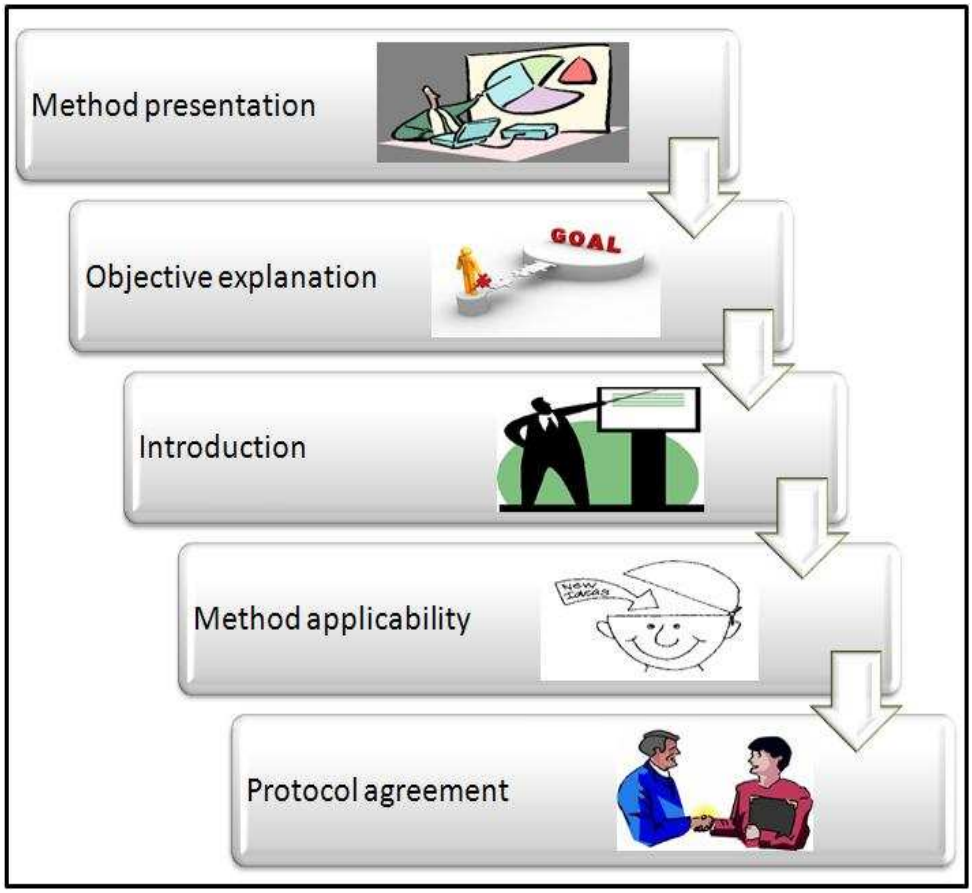

Figure 8-3: Overview of the case study interview

8.4.2 Archival data

Archival data in software engineering can be in the form of meeting minutes, financial records, requirements documents and documents from different development phases [85]

In our case study, and since data collected during interviews can't provide all the information required, features which represent the basic units for release planning and re-planning were elicited using the following documents:

- Tool documentation
- Tool functionalities
- Requirements documents

We realized that part of the data may be missing, because these documents were not originally developed with the intention to provide data for our research. So, we combined the data obtained using archival data analysis with another data collection technique. Instead of conducting surveys and since the resources allocated for this case study was limited, we decided to collect additional information by interacting with the tool.

8.5 Case Study Data

8.5.1 Interview Questions

As the interview wasn't recorded, memos were taken for questions and their answers, as well as further concerns of decision makers, as follows:

1. Release planning and re-planning is usually done ad hoc by one person, the project manager.
2. Features future modification effort is not considered at all when performing baseline release planning and re-planning.
3. Many issues affect the release planning and re-panning such as: the cost of implementing a feature in USD and the expected feature's value to the stakeholder.
4. Considering a feature as a collection of related requirements is reasonable. This is because features are packages of functionalities shipped to the customers. In fact, one of the documents describing the tool, called "Test Coverage Functionality" summarizes functionalities provided by the tool as a group of features, where each feature is decomposed in to a set of requirements.
5. The document "Test Coverage Functionality" can help in gathering information related to features and their requirements when start constructing the OOFeM.

Further information regarding operationalisms may be obtained by careful analysis of other documents such as tool documentation and requirements specifications.

6. Considering features' impact on system modifiability may pay off if the development environment is very volatile and requirements are changing continuously. Again, this need to be investigated by conducting this case study.
7. The most important issue in software release re-planning is to find an efficient and light-weighted method for prioritizing and assigning features to the rest of the release. Also, an important question needs to be answered is when this re-planning method need to be initiated?

8.5.2 System Features

As a result of analysing the documents mentioned in Section 8.4.2, 19 features were identified for the baseline release planning. Table 8-2 provides features and their dependencies. More comprehensive information about the features, including their requirements and operationalisms is provided in Appendix A.

Table 8-2: Features for the baseline release planning

Feature ID	Feature Name	Coupled to	Precedes
f(1)	Basic project specifier		f(2)
f(2)	Extended project specifier		
f(3)	TCV/PRF selector window		
f(4)	Fast TCV file reader	f(3)	
f(5)	Fast PRJ file reader	f(3)	

Feature ID	Feature Name	Coupled to	Precedes
f(6)	Fast source file reader		
f(7)	Source display window	f(6), f(9)	
f(8)	TCV display frame		f(9)
f(9)	File/Probe navigator window		
f(10)	Basic TCV computation	f(6)	f(11)
f(11)	Actions panel TCV arithmetic		
f(12)	Actions Panel Export TCV Arithmetic	f(11)	
f(13)	TCV statistics computation		
f(14)	Actions Panel Report Generation	f(11), f(13)	
f(15)	Java SWING UI implementation		
f(16)	InstallShield Installer		
f(17)	Documentation		
f(18)	Source buffer display management API		
f(19)	Generic Eclipse displays for TCV interaction windows		

8.6 Features Prioritization and Effort Estimation

Stakeholders were asked to prioritize features according to their value, risk and complexity on a nine point scale. While a feature's value and risk were used for performing baseline release planning and re-planning for the next release, complexity was used in estimating features impact on system modifiability as presented in Section 8.8. Also, effort that will be consumed during features implementation was estimated in

USD. Table 8-3 provides the results of features' prioritization rounded to the nearest nine point scale, and effort estimation rounded to the nearest 100 USD.

Table 8-3: Features prioritization and effort estimation results

ID	Value	Risk	Complexity	Effort
f(1)	7	1	1	$6,800
f(2)	4	2	4	$15,000
f(3)	9	5	5	$3,400
f(4)	6	2	3	$3,400
f(5)	6	2	3	$3,400
f(6)	9	2	6	$6,800
f(7)	9	5	5	$15,000
f(8)	5	1	1	$1,400
f(9)	4	4	4	$3,400
f(10)	6	5	6	$5,500
f(11)	3	5	4	$6,800
f(12)	2	4	4	$1,400
f(13)	9	1	2	$3,400
f(14)	9	2	2	$3,400
f(15)	5	7	7	$27,300
f(16)	1	3	2	$2,000
f(17)	5	1	1	$3,400
f(18)	3	6	8	$2,000
f(19)	7	8	9	$30,000

8.7 OOFeM Construction

OOFeM was constructed using ArgoUML tool [1], based on the information gathered during archival data analysis stage, which are related to features, their requirements and operationalisms, as well as their precedence and coupling dependencies. Figure 8-4 shows the OOFeM obtained from the tool. Since space doesn't allow providing all features' information, operationalisms are not presented in this figure. They are assumed to be obtained from Appendix A.

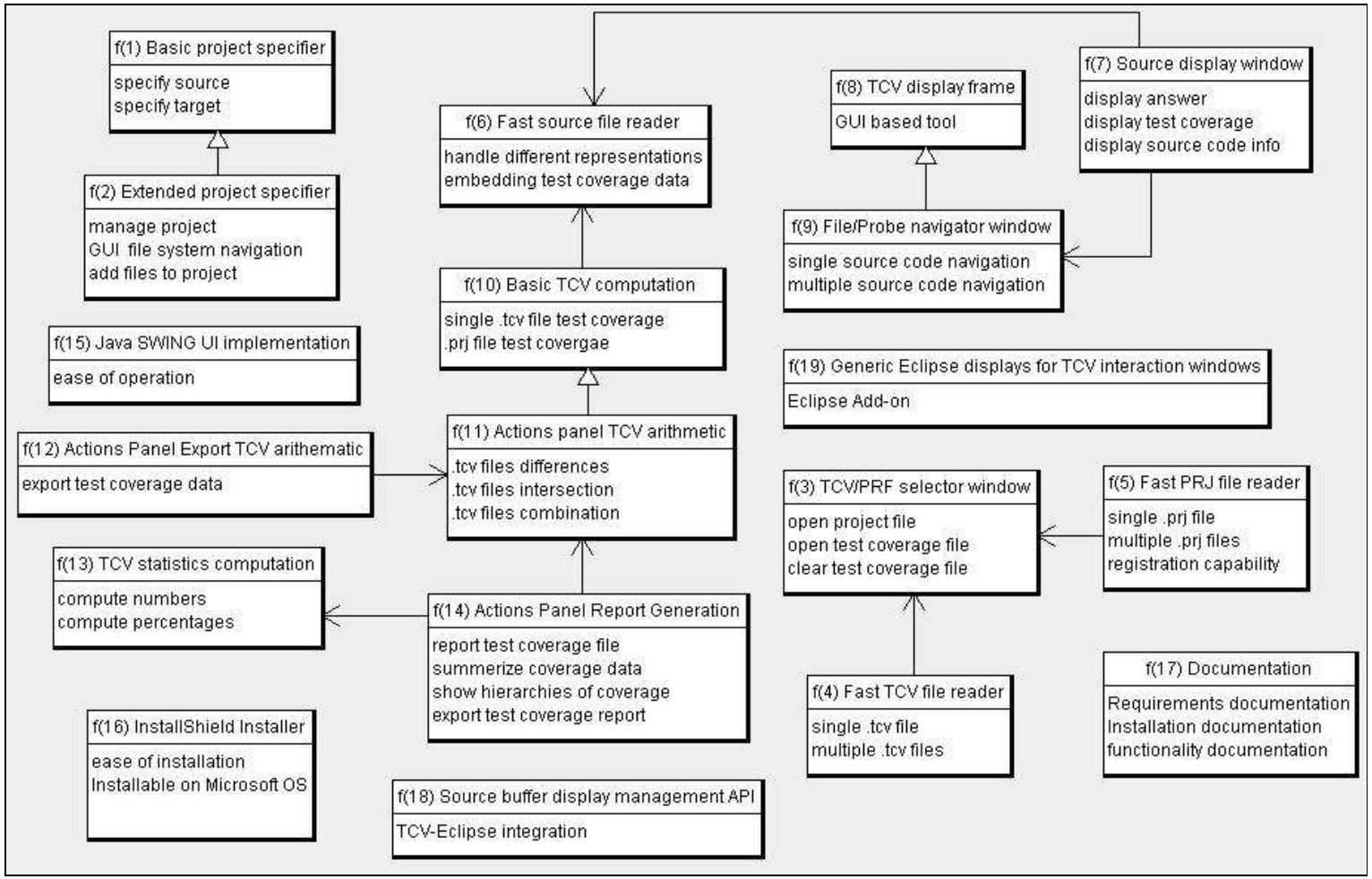

Figure 8-4: OOFeM for system features

8.8 Modifiability estimation

Once OOFeM is constructed, extracting the modifiability information does not require considerable effort. The model obtained from the tool was exported and then imported to the SDMetrics tool [4]. This allows us to partially automate the calculation of the FOMM metrics for the features under consideration. A feature's complexity is assumed to be obtained from the stakeholders during the prioritization process. A screenshot of SDMetrics tool with case study data is shown in Figure 8-5.

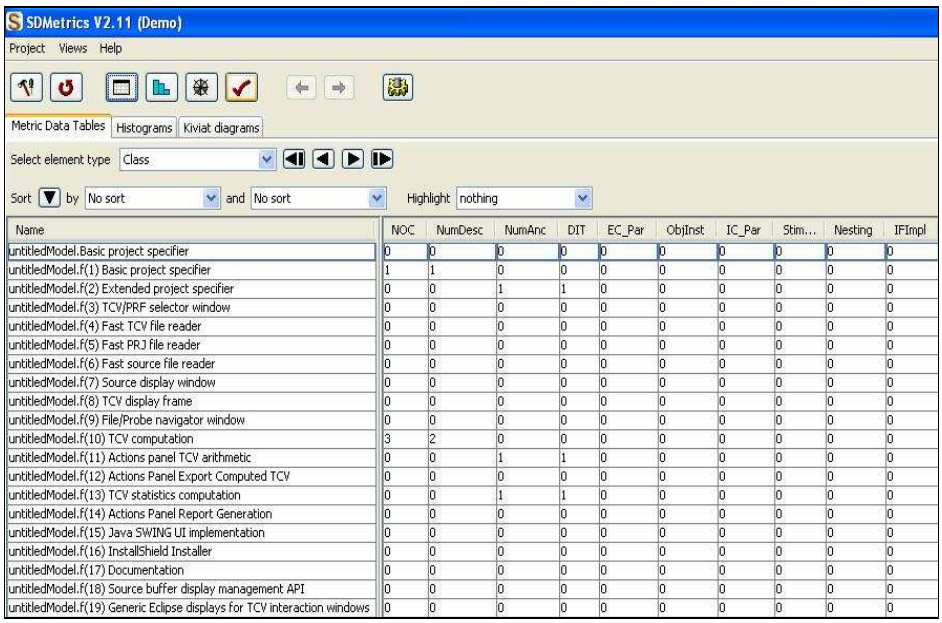

Figure 8-5: Screenshot of the SDMetrics with case study data

After obtaining modifiability metrics from the tool, the effort needed in order to estimate a feature's impact on system modifiability *IMod* becomes reasonable. This is because *IMod* is expressed in terms of a distance to the ideal point. Table 8-4 provides a step-by-step description of how *IMod* was calculated for each feature. First of all, FOMM modifiability metrics are estimated and normalized in the range of [0, 1]. After that, their distances to the ideal point *Dist(n)* was calculated and normalized (i.e., *normDist(n)*).

Finally, as discussed in Chapter 6, features closer to the ideal point are assumed to have the desired modifiability characteristics, and consequently easier to be modified in the future. So, the complement for these normalized distances was estimated (i.e. 1 – *normDist(n)*) and then transformed to a nine point scale in order to be in the range of the attributes used in prioritizing features when planning and re-planning for the next release.

Table 8-4: Features impact on modifiability calculation

ID	*Fcomp (n)*	*norm Fcomp (n)*	*FInh (n)*	*FCoup (n)*	*norm FCoup (n)*	*FCoh (n)*	*Dist (n)*	*normDist (n)*	*IMod (n)*
f1	1	0.11	1	0	0	0.5	1.12	0.84	8
f2	4	0.44	0	0	0	0.33	0.8	0.6	6
f3	5	0.56	0	2	1	0.33	1.33	1	9
f4	3	0.33	0	0	0	0.5	0.6	0.45	5
f5	3	0.33	0	0	0	0.33	0.75	0.56	5
f6	6	0.67	0	2	1	0.5	1.3	0.98	9
f7	5	0.56	0	0	0	0.33	0.87	0.65	6
f8	1	0.11	1	0	0	1	1.01	0.76	7
f9	4	0.44	0	1	0.5	0.5	0.83	0.62	6
f10	6	0.67	1	0	0	0.5	1.3	0.98	9
f11	4	0.44	0	2	1	0.33	1.28	0.96	9
f12	4	0.44	0	0	0	1	0.44	0.33	4
f13	2	0.22	0	1	0.5	0.5	0.74	0.56	5
f14	2	0.22	0	0	0	0.25	0.78	0.59	6
f15	7	0.78	0	0	0	1	0.78	0.59	6
f16	2	0.22	0	0	0	0.5	0.55	0.41	4
f17	1	0.11	0	0	0	0.33	0.68	0.51	5
f18	8	0.89	0	0	0	1	0.89	0.67	6
f19	9	1	0	0	0	1	1	0.75	7

8.9 Release Planning

Once features value, risk, effort and impact on modifiability attributes are available; release planning just applies heuristics to prioritize the features based on these attributes. In order to provide more insight on the applicability of our approach, release planning is done for three different cases as follows:

CASE A: Ad-hoc release planning.

CASE B: Systematic release planning, without addressing the modifiability concern.

CASE C: Systematic release planning, with addressing the modifiability concern.

The following case study setup was applied to all the three cases:

- Planning and re-planning is done just for the next release
- The release under consideration is the first release, R1
- A set of new nineteen features, F_{new} = {f(1),f(2), ..., f(19)} with their corresponding attributes shown in Table 8-2 are considered for release planning.
- Release capacity, CAP = $46,000
- Release start time, T1 = 0
- Release end time, T2 = 46 (i.e. release duration is 46 days)

As described above, the release planning is done for the first release, which means that there is no existing system (i.e. F = Φ). However, this is not the typical situation to show the impact of implementing a feature to the overall system modifiability as part of the dependency is missing. There will be no dependencies between features in F_{new} and existing system features in F. For this reason, Section 8.12 shows how this situation is changed in release R2 by eliciting three new features and showing their impact on the system's modifiability.

8.9.1 CASE A: Ad-hoc Release Planning

In this case, the project manager is performing the release planning without the help of any method. In order to achieve this, stakeholders' feedback as well as features effort and dependencies are all considered by the project manager and used as an input for the release planning. This results in the following two set of features:

- Baseline features, F_b = {f(1), f(6), f(7), f(8), f(9), f(10), f(11) }.
- Rejected features, F_r = {f(2), f(3), f(4), f(5), f(12), f(13), f(14), f(15), f(16), f(17), f(18), f(19)}.

A release value of 43 can be achieved that can be computed by adding up value(n) for all 7 features constitute the baseline release plan as shown in Table 8-5.

Table 8-5: Ad-hoc baseline release plan

Feature	*Effort*	*Acumulated Effort*	*value*
f(1)	$6,800	$6,800.00	7
f(6)	$6,800	$13,600.00	9
f(7)	$15,000	$28,600.00	9
f(8)	$1,400	$30,000.00	5
f(9)	$3,400	$33,400.00	4
f(10)	$5,500	$38,900.00	6
f(11)	$6,800	$45,700.00	3
Total Value			43

8.9.2 CASE B: Systematic Release Planning Without Addressing Modifiability

In this case, we apply the distance to the ideal point and greedy heuristic (discussed in Section 7.3) for release planning. We consider features attributes obtained from stakeholders prioritization such as value, effort and risk, but not the "impact on modifiability". Table 8-6 provides step by step description for features prioritization. As shown in the figure, a feature's distance to the ideal point *Dist* is calculated first. After that, the complement for this distance is calculated after being normalized (i.e. 1-*normDist*). Finally, a feature's priority is estimated by transforming the distance complement from the range of [0,1] to the nine point scale.

Table 8-6: Features prioritization for CASE B

Feature	*Value*	*Risk*	*Effort*	*Dist*	*normDist*	*1 - normDist*	*Priority*
f(13)	9	1	2	1	0.09	0.91	9
f(14)	9	2	2	1.41	0.13	0.87	8.65
f(6)	9	2	3	2.24	0.21	0.79	7.95
f(1)	7	1	3	2.83	0.26	0.74	7.51
f(4)	6	2	2	3.32	0.31	0.69	7.07
f(5)	6	2	2	3.32	0.31	0.69	7.07
f(8)	5	1	1	4	0.37	0.63	6.54
f(17)	5	1	2	4.12	0.38	0.62	6.45
f(3)	9	5	2	4.12	0.38	0.62	6.45
f(10)	6	5	2	5.1	0.47	0.53	5.66
f(9)	4	4	2	5.92	0.55	0.45	4.96
f(7)	9	5	5	5.66	0.52	0.48	5.22
f(2)	4	2	5	6.48	0.6	0.4	4.52
f(12)	2	4	1	7.62	0.7	0.3	3.64
f(11)	3	5	3	7.48	0.69	0.31	3.73
f(18)	3	6	2	7.87	0.73	0.27	3.37
f(16)	1	3	2	8.31	0.77	0.23	3.02
f(15)	5	7	8	10.05	0.93	0.07	1.62
f(19)	7	8	9	10.82	1	0	1

Before applying greedy heuristics to select most attractive features for next release, features dependencies need to be taken in consideration. For this purpose we apply the steps described in Section 7.3.2 in order to group features and update their priorities according to their coupling and precedence dependencies. Table 8-7 provides step by step description on features priorities and effort estimates after considering dependencies.

As shown in the figure, features were sorted at the beginning (Table 8-7-A) according to their priorities. Greedy optimization is applied adding features with high priority to the rest of release. Feature f(13) was identified of the highest priority. Since f(13) doesn't have any dependency with any other feature, it was added to R1. The next feature with highest priority f(14) is coupled to both f(11) and f(13). Since f(11) is not yet included in

R1, both f(11) and f(14) were combined together. Their priority and effort was updated accordingly. Features were sorted again according to their priority after combining f(11) and f(14) (Table 8-7-B). The next highest two features f(1) and f(6) don't have any dependencies. So, they were added to R1. When it comes to f(4) and f(5), and since there is a dependency between these two features and f(3), the three features were combined together (Table 8-7-C). The features were sorted again according to their priority. This resulted in adding f(3), f(4), f(5), f(8), and f(17) to R1. When it came to f(11) + f(14), it appeared that f(11) can't be implemented before f(10). So, the three features (f(10), f(11), and f(14)) were combined again and features were sorted according to their priority. Finally, features don't exceed the release capacity were added to R1, which are: f(9), f(10), f(11), and f(14) (Table 8-7-D).

Since $46,000 was allocated for release R1, we were able to incorporate 12 out of 19 features in the next release baseline plan (i.e. F_b) which are: f(1), f(3), f(4), f(5), f(6), f(8), f(9), f(10), f(11), f(13), f(14), and f(17). On the other hand, 7 features were rejected (i.e. F_r) which are: f(2), f(7), f(12), f(15), f(16), f(18) and f(19).

A release value of 78 can be achieved that can be computed by adding up *value*(n) for all 12 features constitute the release. Also, a total release impact on system modifiability of 81 was obtained that can be computed by adding up *IMod*(n) for all 12 features constitute the release.

Table 8-7: Features priorities and effort estimates after considering dependencies for CASE B

(A)

Features	*Priority*	*Effort*	*Accumlated Effort*
f(13)	9	$3,400	$3,400
f(14)	8.65	$3,400	
f(6)	7.95	$6,800	
f(1)	7.51	$6,800	
f(4)	7.07	$3,400	
f(5)	7.07	$3,400	
f(8)	6.54	$1,400	
f(17)	6.45	$3,400	
f(3)	6.45	$3,400	
f(10)	5.66	$5,500	
f(9)	4.96	$3,400	
f(7)	5.22	$15,000	
f(2)	4.52	$15,000	
f(12)	3.64	$1,400	
f(11)	3.73	$6,800	
f(18)	3.37	$2,000	
f(16)	3.02	$2,000	
f(15)	1.62	$27,300	
f(19)	1	$30,000	

(B)

Features	*Priority*	*Effort*	*Accumlated Effort*
f(13)	9	$3,400	$3,400
f(6)	7.95	$6,800	$10,200
f(1)	7.51	$6,800	$17,000
f(4)	7.07	$3,400	
f(5)	7.07	$3,400	
f(8)	6.54	$1,400	
f(17)	6.45	$3,400	
f(3)	6.45	$3,400	
f(11)+f(14)	6.01	$5,100	
f(10)	5.66	$5,500	
f(9)	5.22	$3,400	
f(7)	4.96	$15,000	
f(2)	4.52	$15,000	
f(12)	3.64	$1,400	
f(18)	3.37	$2,000	
f(16)	3.02	$2,000	
f(15)	1.62	$27,300	
f(19)	1	$30,000	

(C)

Features	*Priority*	*Effort*	*Accumlated Effort*
f(13)	9	$3,400	$3,400
f(6)	7.95	$6,800	$10,200
f(1)	7.51	$6,800	$17,000
f(3)+f(4)+f(5)	6.86	$10,200	$27,200
f(8)	6.54	$1,400	$28,600
f(17)	6.45	$3,400	$32,000
f(11)+f(14)	6.01	$5,100	
f(10)	5.66	$5,500	
f(9)	5.22	$3,400	
f(7)	4.96	$15,000	
f(2)	4.52	$15,000	
f(12)	3.64	$1,400	
f(18)	3.37	$2,000	
f(16)	3.02	$2,000	
f(15)	1.62	$27,300	
f(19)	1	$30,000	

(D)

Features	*Priority*	*Effort*	*Accumlated Effort*
f(13)	9	$3,400	$3,400
f(6)	7.95	$6,800	$10,200
f(1)	7.51	$6,800	$17,000
f(3)+f(4)+f(5)	6.86	$10,200	$27,200
f(8)	6.54	$1,400	$28,600
f(17)	6.45	$3,400	$32,000
f(11)+f(10)+f(14)	5.89	$10,600	$42,600
f(9)	5.22	$3,400	$46,000
f(7)	4.96	$15,000	
f(2)	4.52	$15,000	
f(12)	3.64	$1,400	
f(18)	3.37	$2,000	
f(16)	3.02	$2,000	
f(15)	1.62	$27,300	
f(19)	1	$30,000	

8.9.3 CASE C: Systematic Release Planning by Addressing Modifiability

In this case, we apply the same steps as in CASE B, but with considering the estimated features impact on modifiability.

Table 8-8 provides step by step description for features prioritization according to their distances from the ideal point.

Table 8-8: Features prioritization for CASE C

Feature	*Value*	*Risk*	*Effort*	*IMod*	*Dist*	*normDist*	*1-normDist*	*Priority*
f(14)	9	2	2	5	4.24	0.34	0.66	9
f(4)	6	2	2	5	5.2	0.42	0.58	8.03
f(6)	9	2	3	6	5.48	0.44	0.56	7.79
f(8)	5	1	1	6	6.4	0.52	0.48	6.82
f(5)	6	2	2	6	6	0.49	0.51	7.18
f(10)	6	5	2	5	6.48	0.52	0.48	6.82
f(13)	9	1	2	8	7.07	0.57	0.43	6.21
f(2)	4	2	5	5	7.62	0.62	0.38	5.61
f(12)	2	4	1	4	8.19	0.66	0.34	5.12
f(18)	3	6	2	4	8.43	0.68	0.32	4.88
f(1)	7	1	3	9	8.49	0.69	0.31	4.76
f(17)	4	4	2	7	8.43	0.68	0.32	4.88
f(9)	5	1	2	9	9	0.73	0.27	4.27
f(11)	3	5	3	6	9	0.73	0.27	4.27
f(3)	9	5	2	9	9	0.73	0.27	4.27
f(16)	1	3	2	6	9.7	0.78	0.22	3.67
f(7)	9	5	5	9	9.8	0.79	0.21	3.55
f(15)	5	7	8	6	11.22	0.91	0.09	2.09
f(19)	7	8	9	7	12.37	1	0	1

Again, features dependencies were considered. Table 8-9 shows the final priorities and efforts estimates for features after considering their dependencies. These results were obtained by following the same steps used for CASE B as in Table 8-7.

Since \$46,000 was allocated for release R1, we were able to incorporate 12 out of 19 features in the next release baseline plan (i.e. F_b) which are: f(3), f(4), f(5), f(6), f(8), f(10), f(11), f(12), f(13), f(14), f(17), and f(18). On the other hand, seven features were rejected (i.e. F_r) which are: f(1), f(2), f(7), f(9), f(15), f(16), and f(19).

A release value of 71 can be achieved that can be computed by adding up *value*(n) for all 12 features constitute the release. Also, a total release impact on system modifiability of 71 was obtained that can be computed by adding up *IMod*(n) for all 12 features constitute the release.

Table 8-9: Features priorities and effort estimates after considering dependencies for CASE C

Features Set	*Priority*	*Effort*	*Accumlated Effort*
f(6)	7.79	$6,800	$6,800
f(8)	6.82	$1,400	$8,200
f(10)	6.82	$5,500	$13,700
f(3), f(4), f(5)	6.49	$10,300	$24,000
f(11), f(13), f(14)	6.49	$13,600	$37,600
f(12)	5.12	$1,400	$39,000
f(1), f(2)	5.09	$21,800	
f(18)	4.88	$2,000	$41,000
f(17)	4.88	$3,400	$44,400
f(9)	4.27	$3,400	
f(16)	3.67	$2,000	
f(7)	3.55	$15,000	
f(15)	2.09	$27,300	
f(19)	1	$30,000	

8.10 Change Requests Arrival

We simulate ten new features and change requests (50% of baseline features): f(20), f(21), ..., f(29) are arriving at different points in time within the release duration. The corresponding attributes: value, risk, impact on modifiability, and effort are also randomly generated within the same range (e.g., in a 9 point scale) in the set of features. The release duration for R1 was 46 days (i.e. $1000 per day). The arrival time for the change requests, time(n), is equally distributed over the time period T1 = 0 to T2 = 46. Besides, coupling and precedence dependencies were generated as well. Table 8-10 provides the information about the simulated data.

Table 8-10: Simulated change requests

f(n)	time(n)	value(n)	risk(n)	IMod(n)	effort(n)	coupled	precede
f(20)	3	6	4	5	$2,000		*f(23)*
f(21)	7	2	5	7	$6,600		
f(22)	9	8	3	1	$3,600	*f(20)*	
f(23)	13	1	8	8	$3,400		
f(24)	15	9	1	4	$2,500		*f(27)*
f(25)	21	4	6	2	$3,800	*f(27)*	
f(26)	26	7	2	3	$5,700		
f(27)	32	2	9	6	$4,800		
f(28)	35	5	5	9	$2,600	*f(25)*	
f(29)	39	8	3	6	$4,100		

8.11 Release Re-planning

8.11.1 Step 1: When to Re-plan?

We consider a value-based trigger for re-planning. Once the accumulated *value*(n) of the incoming change requests and new features reaches or exceeds the acceptable amount of change *h*(t) as stated in condition (7.2), the re-planning process is initiated. Therefore, condition (7.2) is continuously checked within the re-planning period of 0 to 46. In our case, the parameter λ was adjusted to 0.4. As soon as the change requests f(24) arrives at time 15, the relative accumulative value exceeds the value of h(t) as shown in Table 8-11.

Table 8-11: Re-planning condition for change requests at differnt point in time

f(n)	time	Value(CR(t)) / Value(F_b)	h(t)	Value(CR(t)) / Value(F_b) > h(t)
f(20)	3	0.078	0.374	NO
f(21)	7	0.104	0.339	NO
f(22)	9	0.208	0.322	NO
f(23)	13	0.221	0.287	NO
f(24)	15	0.338	0.27	YES
f(25)	21	0.39	0.217	NA
f(26)	26	0.481	0.174	NA
f(27)	32	0.506	0.122	NA
f(28)	35	0.571	0.096	NA
f(29)	39	0.675	0.061	NA

8.11.2 Step 2: How to Re-plan

At the second step, and after determining when to re-plan, the question now becomes how to do that. The answer lies in applying the same greedy optimization as performed in the case of baseline release planning. However, this time we consider a new candidate list that includes the change requests, features that are planned but have not been implemented in the release, and the rejected features from the baseline release plan.

At re-planning time, some of the planned features have already been implemented, or in the process of implementation. Features already implemented are excluded from the re-

planning process, while features under progress need to be assessed and evaluated, in order to make a decision whether to keep implementing them (i.e. not consider them for re-planning), or to cancel their implementation (i.e. consider them for re-planning). For this decision in order to be taken, some form of operational planning is required in order to know the sequence of implementing the features and their status at the re-planning time. Table 8-12 shows the sequence of implementing the features in the baseline release plan. The assumption is that features are implemented in the order of their priorities.

As shown in the figure, at the time t = 15, three features are already implemented which are: f(6), f(8), and f(10); while features f(3) is in progress. We follow the description in Section 7.3.3 in order to make a decision regarding whether or not to keep implementing f(3). The assumption is that there is a decisions toward cancelling the implementation of f(3) as it was only 9% completed, which is far less than the pre-defined percentage of completion (75% completed) for which the features is allowed to be implemented if re-planning occurs.

Table 8-12: Baseline operational plan

Feature	start(n)	end(n)	Accumulative effort
f(6)	1	7	$6,800
f(8)	7	9	$8,200
f(10)	9	14	$13,700
f(3)	14	18	$17,100
f(5)	18	21	$20,500
f(4)	21	24	$23,900
f(13)	24	28	$27,300
f(11)	28	35	$34,100
f(14)	35	38	$37,500
f(12)	38	39	$38,900
f(18)	39	41	$40,900
f(17)	41	45	$44,300

At this re-planning point, the release capacity (CAP) already consumed by the implemented features is $14,000. We still have ($46,000 – $14,000) = $32,000 of capacity remaining for the rest of the release period.

Release planning is conducted again, but with the following candidate list of 21 features:

- The remaining nine features: f(3), f(4), f(5), f(11), f(12), f(13), f(14), f(17), and f(18) not yet implemented.
- The five change requests: f(20), f(21), f(22), f(23), and f(24) arriving until t = 15.
- The seven originally rejected features from the baseline plan f(1), f(2), f(7), f(9), f(15), f(16), and f(19).

Table 8-13 provides step by step description for the prioritization of the new set of 21 features according to their distances from the ideal point.

Table 8-13: Features prioritization for re-planning

Feature	*Value*	*Risk*	*Effort*	*IMod*	*Dist*	*normDist*	*Priority*
f(22)	8	3	2	1	3.87	0.28	9
f(24)	9	1	2	4	4.58	0.33	8.44
f(14)	9	2	2	5	5.74	0.41	7.56
f(4)	6	2	2	5	6.48	0.46	7
f(20)	6	4	2	5	7.35	0.52	6.33
f(5)	6	2	2	6	7.28	0.52	6.33
f(13)	9	1	2	8	8.31	0.59	5.56
f(2)	4	2	5	5	8.89	0.63	5.11
f(12)	2	4	1	4	9.06	0.64	5
f(18)	3	6	2	4	9.59	0.68	4.56
f(1)	7	1	3	9	9.75	0.69	4.44
f(17)	4	4	2	7	9.7	0.69	4.44
f(9)	5	1	2	9	10.1	0.72	4.11
f(11)	3	5	3	6	10.3	0.73	4
f(3)	9	5	2	9	10.49	0.75	3.78
f(16)	1	3	2	6	10.63	0.76	3.67
f(7)	9	5	5	9	11.45	0.81	3.11
f(21)	2	5	3	7	11.49	0.82	3
f(15)	5	7	8	6	12.85	0.91	2
f(23)	1	8	2	8	14	1	1
f(19)	7	8	9	7	14.07	1	1

Greedy heuristics was applied. Table 8-14 shows these features ordered according to their updated priorities.

Since $32,000 was allocated for rest of release R1, we were able to incorporate 9 out of the list of 21 features considered for re-planning in the rest of the release, which are: f(3), f(4), f(5), f(11), f(13), f(14), f(20), f(22), and f(24).

A release value of 85 can be achieved that can be computed by adding up value(n) for all 9 features constitute the rest of the release and the already implemented 3 features.

The overall gain in value is 85 – 71 = 14 (20%).

Table 8-14: Features priorities and effort estimates after considering dependencies for the re-planning case

Features Set	*Priority*	*Effort*	*Accumlated Effort*
f(24)	8.44	$2,500	$2,500
f(20)+f(22)	7.67	$5,600	$8,100
f(11)+ f(13)+ f(14)	5.71	$13,600	$21,700
f(3)+f(4)+f(5)	5.7	$10,200	$31,900
f(12)	5		
f(2)	5.11		
f(18)	4.56		
f(1)	4.44		
f(9)	4.44		
f(17)	4.11		
f(16)	3.67		
f(7)	3.11		
f(21)	3		
f(15)	2		
f(23)	1		
f(19)	1		

8.11.3 Step 3: What to Re-plan

In Step 3, we determine the best compromise between gaining additional value and losing the stability of the release from replacing features in the baseline plan with change requests. The two extremes in this respect would be no changes at all (highest degree of stability) and replacing features f(12), f(17), and f(18) with features f(20), f(22), and f(24)

(lowest degree of stability). The results of a more detailed analysis are summarized in Table 8-15.

Based on analysis of the trade-off situation as shown in Figure 8-6, it was recommended that the best compromise would be the replacement of one feature from the baseline. The feature to be replaced is f(12). Its recommended replacement is features f(24), and the added value of this replacement is 7 (i.e. 10% gain in baseline release value).

Table 8-15: Evolution of features replacement for the three features mentioned in Step 2

Number of features eliminated from baseline	Set of features to be replaced	Set of replacing features	Added value
1	{12}	{24}	7
2	{12, 18}	{24, 22}	7 + 5
3	{12, 18, 17}	{24, 22, 20}	7 + 5 + 2

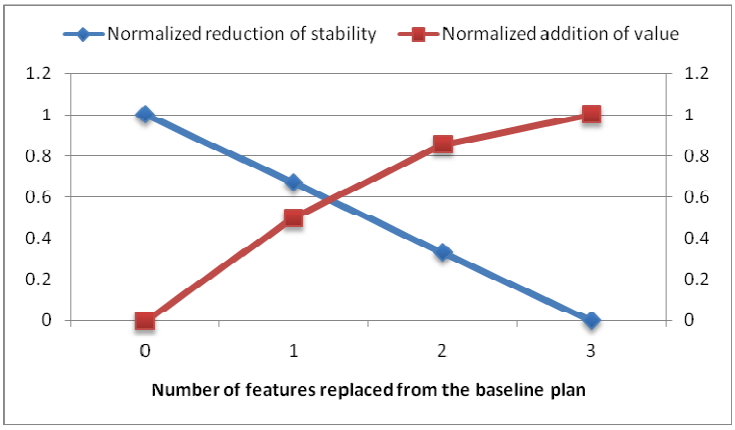

Figure 8-6: Trade-off between the added value and the decrease in baseline release plan stability

8.12 Interacting with Existing System in the Second Release

As mentioned before, there were no dependences between new features and the existing system in the first release. This reduced the scope of the estimated impact of new features to be on the modifiability of the future system, which is composed of the set of features assigned to the first release R1.

In this section we have elicited three new features for the second release R2 (i.e. after implementing the baseline release plan of CASE C), to show how this situation will be changed in the second release. These three new features have dependencies with other features considered in release planning for the second release, as well as already implemented features from the previous release (i.e. the existing system) as follows:

- f(30) “TCV execution frame”: This feature extends f(8) “TCV display frame” by showing different code executions for test coverage.
- f(31) “Eclipse TCV code generator”: This feature uses the test coverage results produced by f(10) “Basic TCV computation” with the functionality provided by f(19) “Generic Eclipse display for TCV interaction window” in order to generate code for Eclipse to solve the problem of unreachable code.
- f(32) “TCV/PRF configuration management”: This is a new requirement added to the system which enforces f(3) “TCV/PRF selector window” to be managed by the settings adjusted in f(32).

Figure 8-7 shows how new features impact the system modifiability as follows:

- f(30) precedence dependency with f(8) increased *FInh* for f(8) and f(30) by 1.
- f(31) coupling dependency with f(10) and f(19) increased *FCoup* for f(10), f(19), and f(31) by 1.

- f(3) coupling dependency with f(32) increased *FCoup* for f(32) by 1.

In all of these cases, the higher the degree of coupling and precedence between features (estimated in terms of *FCoup* and *FInh*), the more effort is required to modify the features in the future.

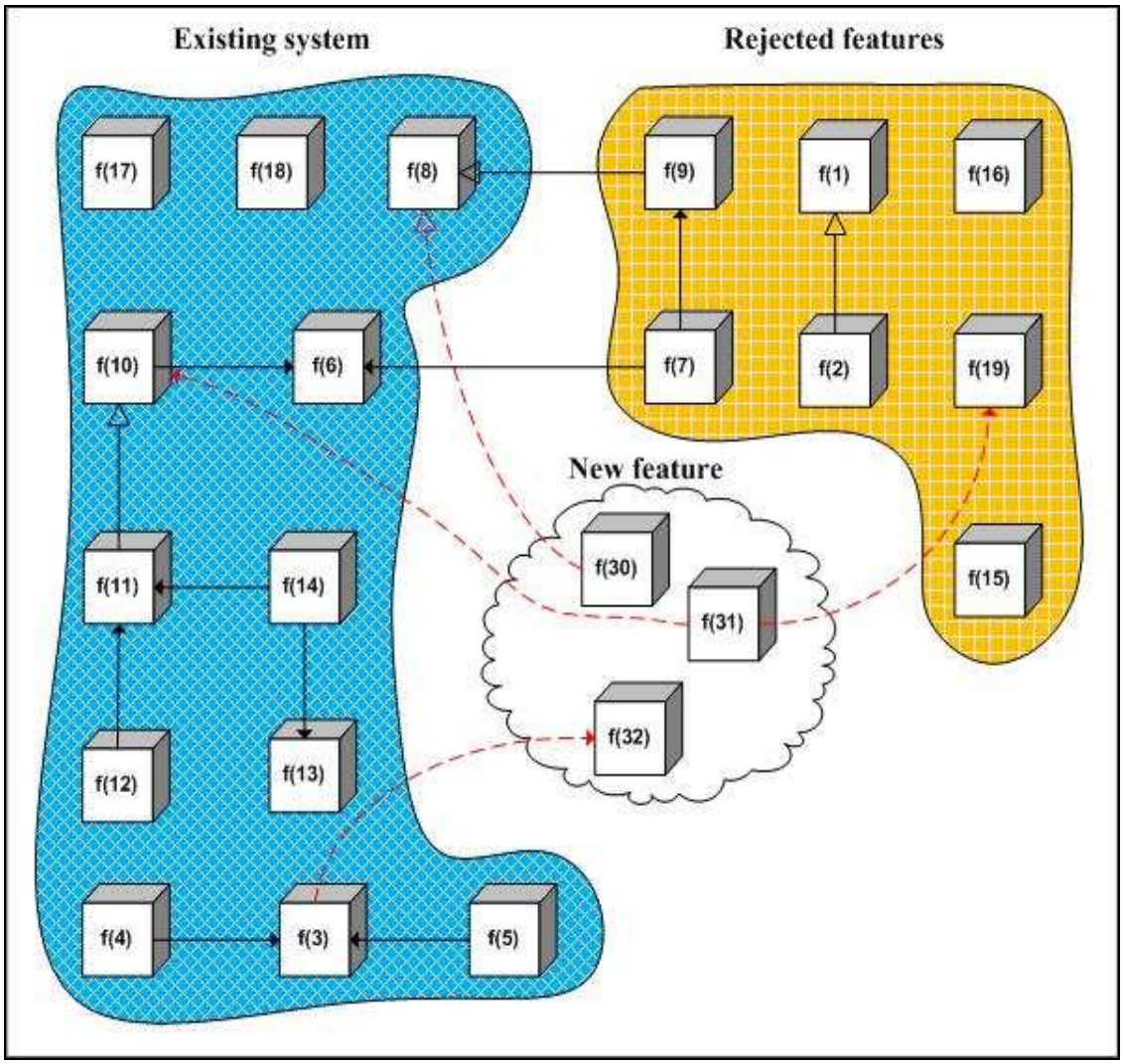

Figure 8-7: Interaction between new features and other parts of the system in Release R2

8.13 Data Analysis

In this section, we perform an analysis based on analysing the results of the interview and archival data analysis, the predicted and actual effort collected after implementing the features, as well as the results of modifiability estimation and release planning.

8.13.1 Interview and Archival data analysis results

We have analyzed the results of the interview and archival data analysis, in order to answer Question 1 which investigates how close our approximation of features in terms of composition and interaction to that in reality, and Question 2 which investigates how applicable is producing and applying our OOFeM approach in reality. We have concluded the following:

- Features dependencies: many types of dependencies between features is considered in reality, such as sequencing, which is similar to the precedence dependency, and overlapping in functionalities which is similar to the coupling dependencies.
- Features composition: refining features as collections of related requirements is done in reality. This is because features are considered as packages of functionalities shipped to the customers. In fact, one of the documents describing the tool, which is called “Test Coverage Functionality” summarizes functionalities provided by the tool as a group of features, where each feature is decomposed in to a set of requirements.
- OOFeM Constructing: constructing OOFeM from archival data analysis required considerable amount of effort. Since features were classified as groups of related requirements, extracting features’ requirements was affordable. However, extracting the operationalisms required careful and thorough analysis of

requirements documents and specifications, which required considerable amount of time.

- FOMM Calculations: Since we are using the ArgoUML and SDMetric tools, the effort required to extract FOMM metrics was reasonable. After constructing OOFeM in ArgoUML, The resulting UML structure was imported into SDMetrics tool which allowed the automatic generation of some of the FOMM metrics.

8.13.2 Impact on Modifiability as an Early Effort Indicator

As discussed in Section 6, a feature's impact on system modifiability is assumed to provide an indication about the expected effort required to modify a feature in the future. The actual effort in USD for implementing and modifying the 19 features in Table 8-2 was collected after few releases. Table 8-16 provides the estimated and actual effort for implementing these features. Highlighted rows indicate a resources deficit in implementation (i.e. actual effort is greater than what was expected).

As show in Table 8-16, five features were found to consume more than additional 50% of the estimated effort, which are: f(3), f(6), f(8), f(9), and f(11). One of the reasons behind this problem was change requests.

In this case study, features' impact on modifiability provided us with warnings on these features which required additional effort. Four out of these five features were identified in Table 8-4 of very high impact on system modifiability (i.e. $IMod \geq 7$), and consequently require more effort in accommodating future changes.

Considering Question 4, which investigates the advantages of addressing the modifiability concern in release planning, we can conclude that considering features impact on systems' modifiability when performing release planning can provide use with

warnings on future effort that may occur as a result of modifying the features. Considering this information at an early point in time may save a lot of effort in the future.

Table 8-16: Estimated and actual effort for implementing features

Feature	*Estimated effort*	*Actual Effort*	*Difference estimated - actual*	*Difference %*
f(6)	$6,800	$10,200	-$3,400	50%
f(11)	$6,800	$10,200	-$3,400	50%
f(3)	$3,400	$5,500	-$2,100	62%
f(9)	$3,400	$5,500	-$2,100	62%
f(1)	$6,800	$8,200	-$1,400	21%
f(13)	$3,400	$4,800	-$1,400	41%
f(8)	$1,400	$2,700	-$1,300	93%
f(18)	$2,000	$2,700	-$700	35%
f(14)	$3,400	$4,100	-$700	21%
f(12)	$1,400	$2,000	-$600	43%
f(4)	$3,400	$3,400	$0	0%
f(16)	$2,000	$2,000	$0	0%
f(17)	$3,400	$3,400	$0	0%
f(5)	$3,400	$2,000	$1,400	41%
f(10)	$5,500	$3,400	$2,100	38%
f(7)	$15,000	$10,200	$4,800	32%
f(19)	$30,000	$23,900	$6,100	20%
f(15)	$27,300	$20,500	$6,800	25%
f(2)	$15,000	not implemented	NA	NA

8.13.3 Release Planning

In this section, we perform an analysis based on the results of the three cases studied in Section 8.9 (CASE A, CASE B, and CASE C). As part of the analysis, Question 3 and Question 4 were considered which investigate the advantages of addressing the modifiability concern and systematic release planning. Answering Question 3 requires a comparison between CASE A and CASE B, whereas answering Question 4 requires a comparison between CASE B and CASE C. After performing these comparisons, we have concluded the following:

- Systematic release planning is as good as or better than ad-hoc release planning: The reason behind this conclusion is that, as a result of applying systematic release planning, we were able to achieve the following:
    - An increase of total release value from 43 in CASE A to 76 in CASE B.
    - An increase of the total number of features from 7 in CASE A to 12 features in CASE B.
    - Addressing the risk concern in release planning, which was not considered at all when performing Ad-hoc release planning.

  This is because the systematic release planning compromises between features value, risk, and effort when performing the prioritization. This allows assigning most valuable and least risky features in the next release. In this same time, it increases the number of features assigned to the next release as it considers the implementation effort in the prioritization process which enables the inclusion of multiple features requiring less effort than one big feature.
- Addressing the modifiability concern in the baseline release planning enabled us to save future modification effort in future at the cost of the baseline release value, as follows:
    - Two features in the baseline plan of CASE B (f(1) and f(9)) were replaced with two other features in CASE C (f(12) and f(18)) which are assumed to be with lower impact on system modifiability. This replacement resulted in saving $2,200 of future modification cost as shown in Table 8-16. This is because implementing f(1) and f(9) required additional $3,500 while implementing f(12) and f(18) required $1,300.

- The total release value went down from 78 in CASE B to 71 in CASE C as a result of adding another criterion for features prioritizing (impact on modifiability).

Although were able to save future modification effort, it is left to the decision maker to decide whether to invest in modifiability estimation and consideration in release planning or not. This depends on the following questions:

- Q1: *Is the environment for which the release planning is performed of high volatility and continuous modifications?*
- Q2: *Is the effort required for modifiability estimation reasonable or not? In other words, Are the documents required for constructing OOFeM available or not?*

8.13.4 Release Re-planning

In this section, we perform an analysis based on the result of CASE C which represents the baseline release plan to be implemented and the re-planned release which represents CASE C after accommodating change requests. We have found that substantial change in the overall value of the plans has occurred as a result of re-planning. In our case, just one feature has been exchanged to achieve about 10% gain in the value. Since the dataset size is very small, any small modification in the release can produce a substantial difference in value. Hence, this modification needs to be done carefully. Again, this reminds us of the importance of the H2W re-planning method (Question 5), which updates the release plan with new requirements, without breaking the stability constraint. This is assumed to increase the stakeholders' satisfaction.

8.13.5 Limitations

The above discussion and analysis is subject to the following limitations:

- The findings obtains from this case study cannot be generalized, and can only be considered as an initial evaluation of the method. More case studies and experiments with various sizes and domains need to be conducted to confirm these findings.
- Although PROSUP may save future modification effort by early addressing of the modifiability concern in release planning, it is important to note that PROSUP requires extra effort that is not always affordable. So, it is left to the decision maker to decide whether to invest in modifiability estimation and consideration in release planning or not.
- The dataset size is very small as there are only 19 features for release planning. and re-planning. Such a very small size influenced our results in addressing modifiability, as follows:
  - The heuristics used to handle coupling and precedence dependencies between features highly influenced our assignment of features to the next release. This can be seen by the similarity between the coupled features assigned to both CASE B and CASE C. Larger dataset size would have minimized the effect of coupling by creating more potentialities for features assignment.
  - The small dataset size resulted in just three features considered for value-stability trade-off analysis in Step3 of re-planning (what to re-plan?). Larger dataset size would have provided the reader with more insight in the trade-off

analysis by providing more combinations between replaced features and change requests.

- The kind of available documents used to elicit features in OOFeM didn't allow facilitating the cohesion attribute between operationalisms in features. Based on the available data, we were not able to identify operationalisms across many requirements, which didn't allow the full utilization of the cohesion property.
- It was hard to collect the initial data required for evaluating PROSUP. This was because of the lack of documentation. Therefore, we had to spend more effort interacting with the tool in order to elicit more information. Some of this information was subjective as the researcher is knowledgeable of the method under investigation.
- There was no existing system in the baseline release planning of CASE C. This is not the typical situation to show the impact of implementing a feature to the overall system modifiability, because part of the dependency with the existing system is missing. Considering release planning for a subsequent release (i.e. R2 or R3) where there are dependences between new features and existing ones is required for more comprehensive evaluation of the method.
- We have assumed that features are implemented in the order of their priority. However, more detailed operational plan, which considers types of resources (i.e. design, development and testing) and their dependencies, may result in completely different operational plan.
- The simulated change requests in the re-planning scenario are used just as a proof of concept. Change requests representing real world modifications and new

requirements are required for more comprehensive evaluation of the re-planning method.

- The greedy optimization applied for both planning and re-planning is known to deliver good, but not necessarily optimal results. It was used to keep the method light-weighted.

Chapter Nine: Conclusion and Future Work

This chapter wraps up the thesis by summarizing the research contributions and findings, as well as the limitations and threats to validity. It ends by providing an outlook for future extension of the proposed research.

9.1 Summary and Contribution

The mail goal of this thesis is to: provide decision support for handling change requests in software release planning. We have further broken this objective into the following four research objectives:

- **Objective 1**: *Design a feature modeling technique that captures features characteristics which are related to their impact on system modifiability.*
- **Objective 2**: *Develop a method for estimating features impact on system modifiability and integrating these results in software release planning.*
- **Objective 3**: *Develop a decisions support method for release re-planning, which handles change requests once they arrive and accommodates them in the release plan without violating release stability constraint.*
- **Objective 4**: *Evaluate the decision support method in order to determine its applicability.*

In the process of achieving these objectives, we have made a number of contributions in the field of software engineering, which can be summarized as follows:

9.1.1 Object Oriented Feature Modeling Technique

We have introduced a modeling technique for features in software release planning, called Object Oriented Features Modeling (OOFeM). OOFeM was adapted from the object oriented domain. It utilizes concepts known from UML structure diagrams, such as

complexity, coupling, inheritance and cohesion, and maps them to the feature domain, in order to capture features modifiability characteristics, and eventually estimate a feature's impact on system modifiability.

9.1.2 Estimating Features Impact on System Modifiability

We have researched and identified the Feature Oriented Modifiability Metrics (FOMM), which are adapted from object orientation and mapped into release planning domain, such as features complexity, coupling, inheritance, and cohesion metrics. These metrics are assumed to estimate a feature's impact on system modifiability, and consequently indicate future modification effort. Besides, we have developed a method for estimating these metrics and integrating them in the release planning. This is assumed to support including features with relatively low future modification effort in the baseline release plan. Hence, providing proactive decision support (PROSUP) to mitigate the impact of change requests before their arrival

9.1.3 H2W Re-planning Method

We have designed and developed a decision support method for release re-planning, called H2W. H2W provides reactive decision support (RESUP) in order to addresses change requests once they arrive and accommodate them in the existing software release. It answers the questions of how, when, and what to re-plan for an existing software release?

- WHEN to re-plan: an approximation for the likelihood to change in software releases as well as a value-based re-planning approach is proposed.

- HOW to re-plan: a greedy heuristic based on prioritization of candidate features, based on their distance to an ideal point was proposed. This finds the best compromise between features' value, effort, risk and impact on modifiability.
- WHAT to re-plan: a trade-off analysis between the degree of change related to the originally announced release plan and the value improvement achieved by replacing existing features with more attractive ones is suggested.

9.1.4 Sup-HCR Decision Support Method

We have packaged and combined the methods and techniques mentioned above into one method, called the Sup-HCR. Sup-HCR provides decision support for accommodating change requests in software release planning in proactive and reactive manner. Proactively, Sup-HCR estimates features impact on system modifiability after representing features using OOFeM, when performing baseline release planning. Hence, provides proactive decision support (PROSUP) in considering features' future modification effort at an early point in time. On the other hand, Sup-HCR provides reactive decision support (RESUP) for accommodating change requests once they arrive, using the H2W method. Compromising between the proactive and reactive aspect of the change requests problem mitigates the impact of change before it occurs, and ensures that proper processes are established up-front to handle such changes in the future (if any).

9.1.5 Empirical Evaluation

We have conducted a real world case study to evaluate the applicability of our Sup-HCR approach. We have found that subset of our representation of features in OOFeM is already considered in reality. For example, coupling and precedence dependencies are already considered. Also, features are represented as a collection of related requirements.

We have also found that our method for estimating features impact on system modifiability can be used as an alarm for expected future modification effort, as 4 out of the 5 features which required more than additional 50% of the estimated effort in the case study, were identified of high expected modification effort early. Finally, we have found that the H2W re-planning method enabled us to update the release with new requirements, while not breaking the stability constraint, and keeping the process light-weighted.

9.2 Limitations and Threads to Validity

The application of Sup-HCR has several technical problems and limitations. There are also limitations with the internal validity of the results. This is caused by simplifications made on the underlying model. Some of these limitations were discussed at the case study chapter. The remaining limitations are discussed as follows:

- There effort required to construct the OOFeM and estimate a feature's impact on system modifiability is considerable. This is because of the additional effort required to analyze and verify many documents in order to elicit features modifiability characteristics.
- The underlying assumption behind this research is that the system under consideration is very volatile, and there is a strong motivation to design the system upfront to accommodate future changes. Therefore, Sup-HCR is only applicable for highly volatile environments expecting continuous modifications. Otherwise, the effort invested in estimating the impact on modifiability may not pay off.

- We aggregate all metrics into one value for the feature impact on system modifiability. However, in reality, different metrics might have different degrees of impact on the modifiability. This would depend on the individual feature and the system under study.
- The impact on system modifiability of the feature itself depends on many factors, such as the type of change, difficulty of modification and the extent of modification [88]. One change might be easier to accommodate than another one. Thus, question-based evaluation techniques for assessing the modifiability of features or the system could supplement the metrics in order to judge the impact of implementing a feature in a particular change scenario on system modifiability.
- The selected metrics aggregated into feature modifiability indicator might not be exhaustive. We have selected two metrics for each of internal and external modifiability characteristics of the features. However, other metrics might impact the modifiability as well, as there are other types of dependencies between features.
- We assume that feature impact on modifiability has an additive impact on system modifiability. In order to approximate the total impact on the modifiability of the system, we sum up individual feature modifiability indicators. However, some features may have stronger impact when considered with other features in the system. The same is also applied when estimating the release value.
- We just consider effort as the attribute characterizing the amount of implementation needed to implement the feature. However, in reality, effort is composed of different types of resources; such as design, development and testing

resources. In a more fine-grained model, different types of resources would need to be considered.

- In re-planning, we have made the simplifying assumption that the features are pursued in the order of priority (features with highest priority are implemented first) with all effort spent on the feature in progress. In other words, we have the model of serial implementation of all the features. This again is a simplification which allows re-planning in a reasonable amount of time and with a reasonable amount of computational effort.
- Finally, to keep the planning and re-planning method reasonably light-weight, we applied greedy heuristics known to deliver good, but not necessarily optimal results. As part of the distance-from-ideal-point minimization, effort estimates were mapped to a nine-point scale when used as part of the distance computation.

9.3 Future Research

Throughout this thesis, we have answered several research questions, yet many more were left. This section lists suggestion for possible extensions of this research work. Future research would be in the following directions:

- More comprehensive empirical and industrial evaluation of the approach, by conducting similar case studies with various projects sizes and domains.
- Further investigation of other feature modeling and representation techniques that may contribute to the prediction of features impact on system modifiability, which can be obtained with less effort.
- Further investigation of other factors that may impact feature's modification effort, such as the type of change, degree of change and difficulty of modification.

- Further investigation of other metrics, which have been proven to impact system modifiability that can be mapped to our domain, in order to estimate a feature's future modification effort.
- Providing decision support for aggregating all factors and metrics used to predict features impact on system modifiability in to one measure, and not just relying on their distances to the ideal point. This varies from project to project and defined by the software architect, taking into account the architecture of the system.
- Considering other types of resources, such as design, implementation and testing resources, in order to model effort in our approach, and not just relying on one combined measure (i.e. effort(n)).
- Providing more comprehensive operational planning in order to know the status of features at the re-planning time. This needs not to be only based on the features priorities, but also on the type of resources considered for effort and their dependencies
- Further enhancing our re-planning condition by modeling further requirements and constraints affect triggering re-planning, such as cumulative re-plans value and organizational circumstances.
- Further integrating the quality concern in the re-planning process, by balancing effort between creating additional functionality, and stabilizing the existing code and fixing detected defects.
- Applying rigorous planning and optimization methods and techniques such as EVOLVEII [82] in order to achieve the optimal results.

References

1. *Argo UML tool, http://argouml.tigris.org.*
2. *IBM Rational Rose, http://www-01.ibm.com/software/awdtools/developer/rose/.*
3. *Release Planner, Expert Decision Inc., www.releaseplanner.com*. 2009.
4. *SDMetrics, The Software Design Metrics tool for the UML, http://www.sdmetrics.com/.*
5. Al-Emran, A., Pfahl, D., and Ruhe, G., *"A method for re-planning of software releases using discrete-event simulation"*. Software Process: Improvement and Practice, 2008. 13(1): p. 19-33.
6. Albourae, T., Ruhe, G., and Moussavi, M. *"Lightweight Replanning of Software Product Releases"*. in *Software Product Management, 2006. IWSPM '06. International Workshop on*. 2006. p. 27-34.
7. Bachmann, F., Bass, L., and Nord, R., *Modifiability Tactics*. 2007, CMU/SEI-2007-TR-002). Software Engineering Institute, Carnegie Mellon University, 2007. http://www. sei. cmu. edu/publications/documents/07. reports/07tr002. html.
8. Bagnall, A.J., Rayward-Smith, V.J., and Whittley, I.M., *"The next release problem"*. Information and Software Technology, 2001. 43(14): p. 883-890.
9. Basili, V.R., Briand, L.C., and Melo, W.L., *"A validation of object-oriented design metrics as quality indicators"*. IEEE Transactions on Software Engineering, 1996. 22(10): p. 751-761.
10. Basili, V.R., Caldiera, G., and Rombach, H.D., *"The goal question metric approach"*, in *Encyclopedia of Software Engineering*. 1994, John Wiley & Sons: New York, NJ. p. 528-532.
11. Bengtsson, P.O., Lassing, N., Bosch, J., and Van Vliet, H., *Analyzing software architectures for modifiability*. 2000, Citeseer.
12. Boehm, B.W., *"Software risk management: principles and practices"*. Software, IEEE, 1991. 8(1): p. 32-41.
13. Bontemps, Y., Heymans, P., Schobbens, P.Y., and Trigaux, J.C. *"Semantics of FODA feature diagrams"*. in *Workshop on Software Variability Management for Product Derivation*. 2004. Boston, MA. p. 48-58.
14. Botterweck, G., Pleuss, A., Polzer, A., and Kowalewski, S. *"Towards feature-driven planning of product-line evolution"*. in *Proceedings of the First International Workshop on Feature-Oriented Software Development*. 2009. Denver, Colorado: ACM. p. 109-116.
15. Briand, L.C., Daly, J.W., and Wüst, J., *"A unified framework for cohesion measurement in object-oriented systems"*. IEEE Transactions on Software Engineering, 1999. 25(1): p. 91-121.
16. Briand, L.C., Daly, J.W., and Wust, J.K., *"A unified framework for coupling measurement in object-oriented systems"*. Software Engineering, IEEE Transactions on, 1999. 25(1): p. 91-121.
17. Brito e Abreu, F. and Melo, W. *"Evaluating the impact of object-oriented design on software quality"*. in *Software Metrics Symposium, 1996., Proceedings of the 3rd International*. 1996. p. 90-99.

18. Brito, F., *The MOOD Metrics Set*, in *ECOOP'95 Workshop on Metrics*. 1995: Aarhus, Denmark.
19. Carlshamre, P., *"Release planning in market-driven software product development: Provoking an understanding"*. Requirements Engineering, 2002. 7(3): p. 139-151.
20. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., and Natt och Dag, J. *"An industrial survey of requirements interdependencies in software product release planning"*. in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. 2001. p. 84-91.
21. Chidamber, S.R. and Kemerer, C.F., *"A metrics suite for object oriented design"*. IEEE Transactions on Software Engineering, 1994. 20(6): p. 476-493.
22. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., *"Introduction to Algorithms"*. 2001: MIT Press, Cambridge, Ma.
23. CSharp test coverage tool, *Semantic Designs Inc*, *http://www.semdesigns.com/*. 2010.
24. Czarnecki, K., Hwan, C., Kim, P., and Kalleberg, K.T. *"Feature Models are Views on Ontologies"*. in *Proceedings of the 10th International on Software Product Line Conference*. 2006: IEEE Computer Society. p. 41-51.
25. Deligiannis, I., Shepperd, M., Roumeliotis, M., and Stamelos, I., *"An empirical investigation of an object-oriented design heuristic for maintainability"*. Journal of Systems and Software, 2003. 65(2): p. 127-139.
26. Eugene, W.K., *"Software requirements"*. 2003, Redmond, Wash.: Microsoft Press.
27. Eugene, W.K., *"Software requirements : practical techniques for gathering and managing requirements throughout the product development cycle"*. 2nd ed. 2003, Redmond, Wash.: Microsoft Press. xix, 516.
28. Fenton, N.E. and Pfleeger, S.L., *"Software Metrics: A Rigorous and Practical Approach"*. 1998: PWS Publishing Co. 656.
29. Ferber, S., Haag, J., and Savolainen, J. *"Feature interaction and dependencies: Modeling features for reengineering a legacy product line"*. in *Proceedings of the 2nd International Conference on Software Product Lines*. 2002. San Diego, CA: Springer. p. 235-256.
30. Fey, D., Fajta, R., and Boros, A. *"Feature modeling: A meta-model to enhance usability and usefulness"*. in *Proceedings of the 2nd International Conference on Software Product Lines*. 2002. San Diego, CA: Springer. p. 198-216.
31. Freeman, R.E. and Reed, D.L., *"Stockholders and Stakeholders: A new perspective on Corporate Governance"*. California Management Review, 1983. 25(3): p. 88-106.
32. Greer, D. and Ruhe, G., *"Software release planning: an evolutionary and iterative approach"*. Information and Software Technology, 2004. 46(4): p. 243-253.
33. Griss, M., Favaro, J., and d'Alessandro, M. *"Integrating feature modeling with the RSEB"*. in *Proceedings of the International Conference on Software Reuse*. 1998. Victoria, BC. p. 76-85.

34. Harrison, R., Counsell, S.J., and Nithi, R.V., *"An evaluation of the MOOD set of object-oriented software metrics"*. IEEE Transactions on Software Engineering, 1998. 24(6): p. 491-496.
35. Ho-Won, J., Seung-Gweon, K., and Chang-Shin, C., *"Measuring software product quality: a survey of ISO/IEC 9126"*. IEEE Software, 2004. 21(5): p. 88-92.
36. Hove, S.E. and Anda, B. *"Experiences from conducting semi-structured interviews in empirical software engineering research"*. in *11th IEEE International Symposium on Software Metrics*. 2005. p. 10-23.
37. Jadallah, A., Al-Emran, A., Moussavi, M., and Ruhe, G. *"The How? When? and What? for the Process of Re-planning for Product Releases"*. in *Proceedings of the International Conference on Software Process: Trustworthy Software Development Processes*. 2009. Vancouver, B. C., Canada: Springer-Verlag. p. 24-37.
38. Jadallah, A., Galster, M., Moussavi, M., and Ruhe, G. *"Balancing value and modifiability when planning for the next release"*. in *Proceedings of the 25th IEEE International Conference on Software Maintenance, ICSM*. 2009. p. 495-498.
39. Jadallah, A., Galster, M., Moussavi, M., and Ruhe, G., *How to Address Modifiability Concerns in the Process of Planning for the Next Release – A Trade-off Method and its Initial Evaluation*. 2009, University of Calgary - Software Engineering Decision Support Laboratory: Calgary.
40. Javed, T., e Maqsood, M., and Durrani, Q.S., *A study to investigate the impact of requirements instability on software defects*, in *ACM SIGSOFT Software Engineering Notes*. 2004, ACM New York, NY, USA. p. 1-7.
41. Jung, H., *"Optimizing value and cost in requirements analysis"*. Software, IEEE, 1998. 15(4): p. 74-78.
42. Kapur, P., Ngo-The, A., Ruhe, G., and Smith, A., *Optimized staffing for product releases and its application at Chartwell Technology*. 2008, John Wiley \& Sons, Inc. p. 365-386.
43. Kellerer, H., Pferschy, U., and Pisinger, D., *"Knapsack problems"*. 2004, Berlin ; New York: Springer. xx, 546.
44. Ken, S., *"The enterprise and scrum"*. 2007: Microsoft Press. 176.
45. Khoshafian, S. and Abnous, R., *"Object orientation : concepts, analysis & design, languages, databases, graphical user interfaces, standards"*. 1995, New York: Wiley. xxii, 504.
46. Kotonya, G. and Sommerville, I., *"Requirements engineering : processes and techniques"*. 1998, New York: John Wiley. xi, 282.
47. Kurt, B. and Ian, S., *"Managing iterative software development projects"*. Addison-Wesley object technology series. 2007, Upper Saddle River, NJ: Addison-Wesley.
48. Lanza, M., Marinescu, R., Ducasse, S., and MyiLibrary., *"Object-oriented metrics in practice using software metrics to characterize, evaluate, and improve the design of object-oriented systems"*. 2006, Berlin ; New York: Springer. xiv, 205.

49. Lee, K., Kang, K.C., and Lee, J. *"Concepts and guidelines of feature modeling for product line software engineering"*. in *Proceedings of the 7th International Conference on Software Reuse*. 2002. Austin, TX: Springer. p. 62-77.
50. Lee, Y. and Chang, K.H. *"Reusability and maintainability metrics for object-oriented software"*. in *38th Annual Southeast Regional Conference*. 2000. Clemson, South Carolina: ACM New York, NY, USA. p. 88-94.
51. Lehman, M.M., *"Software's future: managing evolution"*. Software, IEEE, 1998. 15(1): p. 40-44.
52. Leon, R., Jack, S.H., and A., K.E., *"The practice of planning : strategic, administrative, and operational"*. 1981, New York ; Toronto: Van Nostrand Reinhold Co. xiv, 385.
53. Lethbridge, T., Sim, S.E., and Singer, J., *Studying Software Engineers: Data Collection Techniques for Software Field Studies*. 2005, Kluwer Academic Publishers. p. 311-341.
54. Li, W. and Henry, S., *"Object-oriented metrics that predict maintainability"*. Journal of Systems and Software,, 1993. 23(2): p. 111-122.
55. Loconsole, A. *"Empirical studies on requirement management measures"*. in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. 2004. p. 42-44.
56. Makela, S. and Leppanen, V. *"Client based Object-Oriented Cohesion Metrics"*. in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*. 2007. p. 743-748.
57. Malaiya, Y.K. and Denton, J. *"Requirements volatility and defect density"*. in *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*. 1999. p. 285-294.
58. Martin, A.P., Paul, A., and Professional Development, I., *"Think Proactive: New Insights into Decision-Making"*. 1983: Professional Development Institute.
59. Martin, F. and Kendall, S., *"UML distilled : applying the standard object modeling language"*. 1997, Reading, Mass.: Addison Wesley Longman. xviii, 179.
60. McCall, J.A., Richards, P.K., and Walters, G.F., *Factors in Software Quality*. 1977.
61. McManus, J., *"Risk management in software development projects"*. 2004, Boston: Elsevier.
62. Meilir, P.-J. and L., C.L., *"Fundamentals of object-oriented design in UML"*. 2000, New York-Boston: Dorset House ;Addison-Wesley. 458.
63. Michael, E.B., *"Software Release Methodology"*. 1999: Prentice-Hall, Inc. 240.
64. Morrisey, G.L., Below, P.J., and Acomb, B.L., *"The executive guide to operational planning"*. 1st ed. 1988, San Francisco: Jossey-Bass. xxvii, 130 p.
65. Moshood Omolade, S. and Guenther, R. *"Bi-objective release planning for evolving software systems"*. in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 2007. Dubrovnik, Croatia: ACM. p. 105 - 114.

66. N., C.R., *"Software engineering risk analysis and management"*. 1989, New York: Intertext. xvii, 325.
67. Natt och Dag, J., Regnell, B., Gervasi, V., and Brinkkemper, S., *"A linguistic-engineering approach to large-scale requirements management"*. Software, IEEE, 2005. 22(1): p. 32-39.
68. Ngo-The, A. and Ruhe, G., *"Optimized Resource Allocation for Software Release Planning"*. IEEE Transactions on Software Engineering,, 2009. 35(1): p. 109-123.
69. Ngo-The, A. and Ruhe, G., *"A systematic approach for solving the wicked problem of software release planning"*. Soft Computing, 2007. 12(1): p. 95-108.
70. Northrop, L., *Achieving Product Qualities Through Software Architecture Practices*. 2004, Carnegie Mellon University.
71. Nosek, J.T. and Palvia, P., *"Software maintenance management: changes in the last decade"*. Journal of Software Maintenance: Research and Practice, 1990. 2(3): p. 157-174.
72. Nurmuliani, N., Zowghi, D., and Powell, S. *"Analysis of requirements volatility during software development life cycle"*. in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*. 2004. p. 28-37.
73. OMG, *OMG Unified Modeling Language (OMG UML), Superstructure*. 2009.
74. Paulk, M.C., Weber, C.V., Wise, C.J., and Withey, J.V., *Key practices of the capability maturity model*. 1991, Carnegie Mellon University, Pittsburgh PA, Software Engineering Institute.
75. Pervan, G. and Maimbo, H. *"Designing a case study protocol for application in IS research"*. in *Ninth Pacific Conference on Information Systems*. 2005. p. 1281-1292.
76. Pressman, R.S., *"Software engineering : a practitioner's approach"*. 6th ed. 2005, McGraw-Hill Higher Education: Boston. xxxii, 880.
77. Ramil, J.F. *"Continual resource estimation for evolving software"*. in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*. 2003. p. 289-292.
78. Riebisch, M. *"Towards a more precise definition of feature models"*. in *Workshop on Modelling Variability for Object-Oriented Product Lines*. 2003. Darmstadt: BookOnDemand Publ. Co. p. 64-76.
79. Riebisch, M., Bollert, K., Streitferdt, D., and Philippow, I. *"Extending feature diagrams with UML multiplicities"*. in *Proceedings of the 6th Conference on Integrated Design and Process Technology*. 2002. Pasadena, CA. p. 1-7.
80. Robson, C., *Real World Research. 2002*, Blackwell Publishing.
81. Rowe, G. and Wright, G., *"The Delphi technique as a forecasting tool: issues and analysis"*. International Journal for Forecasting, 1999. 15: p. 353-375.
82. Ruhe, G., *"Product Release Planning - Methods, Tools and Applications"*. 2010, CRC Press (available June 2010).
83. Ruhe, G., *"Software Release Planning"*, in *Handbook of Software Engineering and Knowledge Engineering*. 2005, World Scientific. p. 365-394.
84. Ruhe, G. and Saliu, M.O., *"The art and science of software release planning"*. IEEE Software, 2005. 22(6): p. 47-53.

85. Runeson, P. and Host, M., *"Guidelines for conducting and reporting case study research in software engineering"*. Empirical Software Engineering, 2009. 14(2): p. 131-164.
86. Saaty Thomas, L., *"Decision making for leaders : the analytic hierarchy process for decisions in a complex world"*. New ed. 2001, Pittsburgh, PA: RWS Publications. xii, 315.
87. Saliu, O. and Ruhe, G. *"Bi-objective release planning for evolving software systems"*. in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 2007. Dubrovnik, Croatia: ACM. p. 105 - 114.
88. Saliu, O. and Ruhe, G., *"Software release planning for evolving systems"*. Innovations in Systems and Software Engineering, 2005. 1(2): p. 189-204.
89. Saliu, O. and Ruhe, G. *"Supporting Software Release Planning Decisions for Evolving Systems"*. in *Software Engineering Workshop, 2005. 29th Annual IEEE/NASA*. 2005. p. 14-26.
90. Sawyer, P., Sommerville, I., Kotonya, G., and Lancaster, U.K. *"Improving market-driven RE processes"*. in *International Conference on Product Focused Software Process Improvement*. 1999. Oulu, Finland. p. 222.
91. Sharp, H., Finkelstein, A., and Galal, G. *"Stakeholder identification in the requirements engineering process"*. in *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on*. 1999. p. 387-391.
92. Solingen, R.V. and Berghout, E., *"The goal/question/metric method : a practical guide for quality improvement of software development"*. 1999, London ; Chicago: McGraw-Hill. x, 199 p.
93. Stark, G.E. *"Measurements for managing software maintenance"*. in *Software Maintenance 1996, Proceedings., International Conference on*. 1996. p. 152-161.
94. Stark, G.E., Oman, P., Skillicorn, A., and Ameele, A., *"An examination of the effects of requirements changes on software maintenance releases"*. Journal of Software Maintenance: Research and Practice, 1999. 11(5): p. 293-309.
95. Steuer, R.E., *"Multiple Criteria Optimization: Theory, Computation, and Application"*. 1986, New York, NJ: John Wiley & Sons.
96. Turner, C.R., Fuggetta, A., Lavazza, L., and Wolf, A.L., *"A conceptual basis for feature engineering"*. Journal of Systems and Software, 1999. 49(1): p. 3-15.
97. Van den Akker, M., Brinkkemper, S., Diepen, G., and Versendaal, J., *"Software product release planning through optimization and what-if analysis"*. Information and Software Technology, 2008. 50(1-2): p. 101-111.
98. Van Lamsweerde, A. *"Requirements engineering in the year 00: a research perspective"*. in *International Conference on Software Engineering ICSE2000*. 2000. p. 5-19.
99. Wand, Y. and Weber, R., *"An Ontological Evaluation of Systems Analysis and Design Methods"*, in *Information Systems Concepts*. 1989, Elsevier Science: Amsterdam.

100. Zhang, W., Mei, H., and Zhao, H., *"Feature-driven requirement dependency analysis and high-level software design"*. Requirements Engineering, 2006. 11(3): p. 205-220.
101. Zowghi, D. and Nurmuliani, N. *"A study of the impact of requirements volatility on software project performance"*. in *Software Engineering Conference, 2002. Ninth Asia-Pacific*. 2002. p. 3-11.

Appendix A: Release Planning Features, Requirements and Operationalisms

f(n)	Feature Name	Requirements	Operationalisms
f(1)	Basic project specifier	specify source	specify source file
			specify source directory
		specify target	specify probe target file
			specify target directory
f(2)	Extended project specifier	manage project	load project
			save project
			save project as
			launch project
			exit project
		GUI file system navigation	show recent document
			show desktop
			show my document
			show my computer
			show network places
		add files to project	recursive add
			non-recursive add
			specify file name
			specify file type
f(3)	TCV/PRF selector window	open project file	open single .prj file
			open multiple .prj files

f(n)	Feature Name	Requirements	Operationalisms
		open test coverage file	open single .tcv file
			open multiple .tcv files
		clear test coverage file	clear single .tcv file
			clear entire .tcv workspace
f(4)	Fast TCV file reader	single .tcv file	read single .tcv file
			determine .tcv files for a .prj file
		multiple .tcv files	read multiple .tcv files
			combine multiple .tcv files
			determine .tcv files for a .prj file
			able to combine thousands of .tcv files
f(5)	Fast PRJ file reader	single .prj file	read single .prj file
			determine if .prj source code has changed
		multiple .prj files	read multiple .prj files
			combine multiple .prj files with same lang.
			combine multiple .prj files with different lang.
			combine multiple .prj files with different dialects
			combined .prj file have projects .tcv file
			able to combine thousands of .prj files
		registration capability	lock down to single workstation
			limit source languages displayed

f(n)	Feature Name	Requirements	Operationalisms
f(6)	Fast source file reader	handle different representations	handle different character set representations
			handle different control character interpretation
		embedding test coverage data	add probes to the source code files
			save the modified files for test
f(7)	Source display window	display answer	answer is easy to understand
			answer is easy to navigate
		display test coverage	Coloured display of test coverage
			Coloured display covers start/end of probe block
			superimpose coverage data on source code
			display multiple intervals per line for multiple probes
		display source code info	display source line number
			display different source code blocks
			display if source code has changes since compilation
f(8)	TCV display frame	GUI based tool	enable visual interaction with source code
			enable visual interaction with test coverage data
f(9)	File/Probe	single source	ease of source code navigation

f(n)	Feature Name	Requirements	Operationalisms
	navigator window	code navigation	simple GOTO specific source code line
		multiple source code navigation	ease of source file selection
			ease of source file's blocks navigation
f(10)	Basic TCV computation	single .tcv file test coverage	find covered blocks in a single .tcv file
			find uncovered blocks in a single .tcv file
		.prj file test covergae	find covered blocks in all .tcv files
			find uncovered blocks in all .tcv files
f(11)	Actions panel TCV arithmetic	.tcv files differences	compute differences bet. .tcv files
			show differences bet .tcv files for specific run
		.tcv files intersection	compute intersection bet. .tcv files
			show the common bet .tcv files for specific run
		.tcv files combination	compute the union of .tcv files
			show the combined results of .tcv files for specific run
f(12)	Actions Panel Export Arithmetic TCV	export test coverage data	export single .tcv file data
			export entire .prj file and its associated .tcv files
f(13)	TCV statistics	Compute	Compute number of probes inserted

f(n)	Feature Name	Requirements	Operationalisms
	computation	numbers	Compute number of covered probes
			Compute number of uncovered probes
		Compute percentages	Compute percentage of covered probes
			Compute percentage of uncovered probes
f(14)	Actions Panel Report Generation	report test coverage file	show probe reference file path
			show test coverage vector file path
			show source code files path
		summarize coverage data	show total number of probes
			show total number of files
			show percentage of covered code
			show percentage of uncovered code
		show hierarchies of coverage	show percentage of coverage per namespace
			show percentage of coverage per package
			show percentage of coverage per class
			show percentage of coverage per method
		export test coverage report	export report as .txt file
			export report as .xml file
f(15)	Java SWING UI implementation	ease of operation	requiring just Windows and Java
			doesn't require users to download other software

f(n)	Feature Name	Requirements	Operationalisms
f(16)	InstallShield Installer	ease of installation	guided installation process
		Installable on Microsoft OS	Installable on Windows 2000
			Installable on Windows XP
			Installable on Windows Vista
			Installable on Windows 7
f(17)	Documentation	Requirements documentation	documented HW requirements
			documented SW requirements
		Installation documentation	documented installation steps
		functionality documentation	documented features
			documented operation
f(18)	Source buffer display management API	TCV-Eclipse integration	must run with standard Eclipse
			must run with IBM IDE Eclipse/RDs
f(19)	Generic Eclipse displays for TCV interaction windows	Eclipse Add-on	must run with standard Eclipse
			must run with IBM IDE Eclipse/RDs