2015-01-28

When-to-release Planning in Consideration of Technical Debt

Ho, Trong Tan

UNIVERSITY OF CALGARY

When-to-release Planning in Consideration of Technical Debt

by

T. T. Ho

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

JANUARY, 2015

Abstract

Shortening release duration is a key element for creating competitive product from the iterative software development process. However, short-term expedients (e.g. code compromises, delayed change requests, etc.) can have long term effects on the future of the application. Such consequences are categorized as "technical debt", on which "interest" (e.g. delays in implementation on sub-optimal code) have to be paid as long as the "principle" (e.g. flaws in design, documentation and implementation) is not refactored and restructured.

There are multifaceted factors that influenced the effective management of technical debt in organizations. In this research, we formulate the concept of debt in the context of (software) product releases, specifically when-to-release decisions. The potential competitive advantage through faster delivery needs to be balanced against the degree of readiness of the product, evaluated based on features business values and systematic testing, and the potentially incurred technical debt. Pro-active analysis of the estimated impact of running through various release scenarios is expected to provide insights and essential inputs for actual decision-making process. This research also evaluates a set of metrics to track, maintain, and potentially reduce the types of debt created during the lifespan of the product, both as part of industry-driven case studies and quantitative evaluation using a prototype tool.

Publications

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

Referred published and accepted papers

[1] J. Ho, and G. Ruhe, "Releasing Sooner or Later: An Optimization Approach and Its Case Study Evaluation", in *Proceedings Workshop RELENG on Release Engineering at ICSE*, 2013.

[2] J. Ho, S. Shahnewaz, and G. Ruhe, "Releasing Sooner or Later: An Optimization Approach and Its Case Study Evaluation", in *2nd International Workshop on Release Engineering (RELENG)*, 2014.

[3] G. Ruhe, and J. Ho, "Release Planning and Technical debt", in *5th International Workshop on Management of Technical Debt*, *Baltimore, MD*, 2013.

[4] M. Felderer, A. Beer, J. Ho, and G. Ruhe, "Industrial evaluation of the impact of quality-driven release planning", in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 62. ACM, 2014.

[5] J. Ho, G. Ruhe, "When-to-release decisions in consideration of technical debt", in *6th International Workshop on Management of Technical Debt*, 2014.

Technical Reports and Poster presentation:

[1] J. Ho, and G. Ruhe, "Releasing Sooner or Later: An Optimization Approach and Its Case Study Evaluation", in *CPSC Industry Day*, 2013, [Online] https://sites.google.com/site/trongtanho/research

[2] J. Ho, S. Shahnewaz, and G. Ruhe, "Decide When-to-release Products with Measurable, and Manageable Technical Debt", in *CPSC Industry Day*, 2014, [Online] https://sites.google.com/site/trongtanho/research

[3] J. Ho, S. M. Didar-Al-Alam, R. Karim, G. Ruhe, and Z. Shakeri, “Technical debt decisions in testing with cost/benefit optimization”, *a case study conducted at SMART Inc.*, 2014.

Acknowledgements

I would like to open this thesis with the acknowledgement and sincere appreciation to the people who have helped make it possible. Without them and their tireless support, there would not be any result of my personal growth and research work presented in this thesis.

- My deepest gratitude to Dr. Guenther Ruhe, my supervisor, for his unwavering guidance, supervision, and support during the course of my Master program. Without him, I would have not even considered pursuing a Graduate program.
- The University of Calgary, and in particular the Department of Computer Science, for having such high academic standards, and a rich pool of resources and library from which I have drawn from.
- All my friends and fellow researchers from the Software Engineering Decision Support Laboratory, and Ms Kornelia Streb, for their intellectual discussions, challenges, collaborations and support.
- My dad and my mom, who taught me that even a small-town boy from Vietnam, education could open doors of opportunities and knowledge that transcend borders and limitations.
- My brother, Chester Ho, for being the amazing young man that he is.
- My best friends, Oren Raz and Kelly Chan, for being there with me in the darkest hours of self-doubt and disorientation.
- My friends and former employers, Elise Furman and Roy Furman, for their patience, encouragement, guidance and books recommendation.
- All of my friends, acquaintances, and people I have met that I have learned things from, with *metta*.

To the small-town Vietnamese boy who dreamed.

Table of Contents

List of Figures

List of Tables

List of Symbols, Abbreviations and Nomenclature

IDE	Integrated development environment
GQM	Goal-Questions-Metrics
MVC	Model-View-Controller
OSS	Open-source software
RP	ReleasePlanner
RPP	Release-planning Problem
TD	Technical debt
TDD	Test-driven development
W2RP	When-to-release planning
W2RP-TD	When-to-release Planning in consideration of technical debt

Chapter One: INTRODUCTION

1. Motivations

The "technical debt" metaphor was first used by Ward Cunningham [24] to refer to long term consequences, such as software decay or difficulty in maintenance, or shortcuts taken during design and implementation of software. With the adoption and scaling of Agile techniques [59], technical debt has also grown, and has yet to be managed effectively [14]. One of the major root causes of incurred technical debt is business pressure on delivering the software faster [1]. In industry surveys [37], CFOs identified future technology focus as consolidating enterprise's software, modernizing applications and utilizing analytic-based decision support tools. Technical debt makes this goal harder (or impossible) as teams have to spend more development time and effort working on non-optimized, difficult to maintain codes or aging components [80]. In a report of technical trends in 2014, it is presented that a line of code averaged to about $3.61 in technical debt, and about $312 billion is spent annually on software debugging [15].

This research aims to better understand the types of technical debt in real world product releases, including strategic and non-strategic debt [39], and different attributes of technical debt, such as visibility, controllability, size, value, and cost [14] [25] [58]. The research, based on previous work and proposed methodology, will formulate technical debt both quantitatively and qualitatively. Based on this set of well-established factors of technical debt, we propose an approach to monitor and manage technical debt in the context of software readiness [78] and when-to-release decisions [48].

The when-to-release decision is an extension of the problem of release planning. In [61], Ruhe has established the problem of what-to-release as a multi-objective optimization between different stakeholders and criteria within products portfolio. Further studies and investigation were done into releasing the best features, in the shortest

time frame pre-determined by business strategy. The goal of this thesis is to identify the decision factors for managing debt [88], directly related to releasing software when it is 'ready' [90], both in term of satisfying previously defined business requirements and quality-reliability measures. In addition to software process engineering improvements, effective technical debt management can potentially create additional, real, and immediate financial benefits to organizations [23]. This interactive, analytic-based approach to when-to-release decisions will allow product managers to minimize negative impact of non-strategic, indivertible technical debt, and maximize the value of strategic, valuable technical debt.

Lastly, the thesis also presents an approach, coupled with preliminary case studies, and prototype tool implementation to improve the debt management in real world businesses and organizations. The purpose of our proposed framework is to be explicit and precise in our measure of technical debt, yet universal enough to utilize the breakthroughs achieved in previous works. The prototype developed aims at being easily adapted and customized into industry-oriented organizations beyond the scope of this research.

2. Background

2.1. Release planning for iterative product development

(Software) Product release planning addresses the challenge of deciding which (set of) features should be offered to the customers and in what order [82]. Features are defined as a set of logically related requirements and functionality that enables the users to successfully perform business objectives. In other words, a feature is an abstraction of business and functionality requirements [96]. For instance, in Finance and Billing applications, double layer password protection is an offered feature. This feature may consist of many sub-requirements. First of all, user should be able to log in to the first layer of security using a username and password that were previously set by them. Then, a

secure text message or number should be sent to the user. Alternatively, a previously set security question can be asked. User needs to key in the correctly provided keywords or answers to continue with the log in. This feature has multiple steps and requirements. In reality, features can be simple or complex, depending on the nature of the software application or product being offered.

In this research, we assume that features are mostly well-defined in the context of product and project release. The relationship between feature modeling and requirement gathering is already formed [82]. It can be established from the above inter-relation between features and requirements that part of the release planning problem is closely related to requirements engineering [96]. In the context of technical debt, building on inaccurate requirements is already incurring future debt on the development teams. Providing the right set of features in the right releases not only increase the business value of the products, but also minimize the future implications of unmanageable technical debt and excessive defects or change request [15] [33].

Iterative and incremental software development is widely practiced in large-scale, complex technical products and projects, often across multiple teams or developers. Iterative development allows project teams to work collaboratively and deliver features to their business users in versions (i.e. releases). In iterative development, functionalities are treated as mini-project, which comprise a set of (interrelated activities) and are delivered incrementally and periodically [6]. Synonymously, the development cycle is broken up into multiple periods (i.e. releases). Each period could be a major release, for example, “Release 1.0”, “Release 2.0”; or a minor release, such as, “Release 1.1”, “Release 1.2”, etc. During each release period, a subset of features is developed and released. The goal of release planning then is to decide which features should be offered in which release in such a way that (i) the plan is realistic, given the various constraints of the project and the

dependencies among features themselves, and (ii) the assignments maximize value to the stakeholders or customer [77].

Release planning has been identified as a "wicked problem" [20], because the problem is inherently and cognitively complex, and can only be modeled approximately. As such, there is a multitude of release planning models [82] in which the problem is modeled to accommodate various business goals at various levels of detail. Some of these models, together with their prototype implementation and state-of-application will be further discussed in Chapter Two:, related works.

2.2. When-to-release planning(W2RP)

While what-to-release problem is mainly concerned with what features to be assigned into which release, the when-to-release problem is concerned with deciding a milestone (or a strategic date) when the next immediate version should be released. The when-to-release problem does not only concern with features' business values, but also the degree of readiness of the version to be released. Release readiness is studied extensively in the setting of software metrics, software reliability and software quality [78] [90].

When-to-release is not often discussed in literature, and is not often tool supported [48]. In classic release planning model, such as the waterfall model, release date are pre-determined based on business needs [12]. In iterative software development, such as Agile methodology (using Scrum based techniques [27]) release dates are pre-determined based on the release cycle or sprints. Release time is often optimized against the reliability and cost of the software [72], rather than business values. Ruhe and McElroy [69] have previously modeled when-to-release using a time-sensitive value measure.

In the context of technical debt, releasing early to meet time-to-market and business pressure has been identified as one of the major root causes of growing technical debt in organizations [1]. Furthermore, with iterative and incremental development, teams

implement features in short cycles of time, adding features in each version, long term's optimality is often overlooked by short term expediencies [27]. Deciding when to release a version, or a product, in consideration of release readiness, business values, and technical debt will be the main focus of this research.

Lastly, managing technical debt is an integral part of risk management in organization. On the other hand, when-to-release decisions are closely related to the business ROI and opportunity that releasing a product may bring. As risk and opportunity are a duality that co-exist in every project, and the unpredictable nature of large-scale project, simulation and prediction model, such as the Monte Carlos simulation proposed by M. Cantor [18] were also studied as part of this research.

2.3. Technical debt (TD)

Technical debt was first established as a metaphor by Ward Cunningham [24] to refer to the long term consequences (e.g. maintenance issues, security issues, etc.) of short term expedients (e.g. code shortcuts, skipped test cases, etc.) during the design and implementation of software products. Similar to financial debt, there are benefits to technical debt, such as faster time-to-market, increased competitive advantage, and preserved human and capital resources [68]. However, just like financial debt, if technical debt is not effectively managed and repaid eventually, it can lead to a cycle of continuously delayed delivery, poorly maintained code, even halt of operations. Such catastrophic event is similar to that of technical bankruptcy [89]. Furthermore, in a more common situation, industry organizations' managers have identified technical debt as "something [they] faced everyday", making the code harder to maintain, and implementing new features even more challenging [62].

In a survey conducted by Cleveland and Ellis [22], it is found that 46% of software developers do not perform thorough testing due to lack of time, and 36% do not believe their companies perform enough pre-release testing. Furthermore, over 60% of

organizations discovered major software errors in production. In [76], the authors established that during software development, any activities that do not produce value for the customer, such as partially completed work, overhead processes, under-utilized features, task switching and defects are considered to be "waste". We consider all these activities as part of technical debt. However, technical debt should not be mistaken for careless defects, "bad coding practice", or "lack of process", as highlighted in [58].

There are mainly two types of debt. The first type of debt is unintentional and nonstrategic, such as unforeseen coupling between functionality, and software deterioration over time. The second type of debt is strategic, often in the form of optimization for the present, for the short term or long term in the development process. Technical debt may be visible or invisible at different stages of this process. In [16], F. Buschman stated that the technical debt metaphor is the most valuable when it is applied for strategic, intentional conscious decisions to simplify development process to achieve short term business goals, and to take on debt in the future releases, through additional maintenance or refactoring efforts. In this categorization, the synergy between technical debt and release planning is significant, as product managers, through predictive release planning analysis, can then decide which types of debt to take on, and which to be refactored in the current release.

Although there are many studies that analyze the impact of technical debt, business users are often not aware of the current debt portfolio in their organizations, and the impact of such technical debt on systems [68]. Letouzey and Coq [60] have proposed the SQALE method to quantify technical debt, in term of line of codes (LOC) that are not optimized, and in efforts required to bring current code to desired code level of conformance. Technical debt, in the SQALE context, is broken down into 8 different indices. However, technical debt management techniques so far focus almost exclusively in design and testing during operational planning alone. In [88], C. Seaman et al. outlined the

importance of technical debt data in decision making process, via the Cost-Benefits analysis and maintaining a meaningful portfolio of debt data in organizations. Understanding, and formulating technical debt, both qualitatively and quantitatively, and create meaningful portfolio-oriented management techniques, in relation to strategic release planning will be examined in this thesis.

The nature of technical debt and when-to-release are both uncertain and inherently complex. By applying estimation techniques, we aim to potentially measure and evaluate both of these dimensions in relative to the effort required. With the understanding of how much effort would be required to reduce the negative impact of technical debt, we can perform analysis to vary the when-to-release time to achieve the most desirable debt profile. Software development cost estimation is the initial assessment of the total resource consumption required to complete a software project or release. Most of the time, the cost of resource in software development is measured in person effort required to implement the project. The Constructive Cost Model (COCOMO), as proposed by Barry Boehm [9], is one of the most common methods studied and adopted in the industry, proven to be successful at effectively and consistently predicting software development cost [50]. This is just the starting point of our estimation technique and not the focus of the thesis. In the event organizations have existing estimation methods and procedures in place, decisions support for when-to-release planning in consideration of technical debt would still be applicable.

2.4. Decision support systems

Decision support systems are computer-based system that assists decision-makers in making informed, analytic based decisions based on previously identified criteria and rationale. This is achieved through "formulation of alternatives, analysis of their impacts, and interpretation and selection of appropriate options for implementation" [85]. Studies about the designs and architecture of decisions support system are well established [17]

[91]. In this research, we examine software decision support tools, especially decisions related to release planning, what-to-release, when-to-release, and monitoring of the next-release problem [82]. There are many decision support systems have been developed, based on the transition between software empirical study, software measurement and process simulation in software engineering [83]. A software engineering decision support system should provide a workflow, a methodology for analyzing available (and complex) data, evaluating, prioritizing based on a set of predetermined criteria and rationale, and generate near-optimal recommendations for human experts to make the final informed decisions [83]. *Figure 1* below provides an example of such system, ReleasePlanner™ [82], developed based on researches in release planning studies from Software Engineering Decision Support Labs (SEDS), at the University of Calgary.

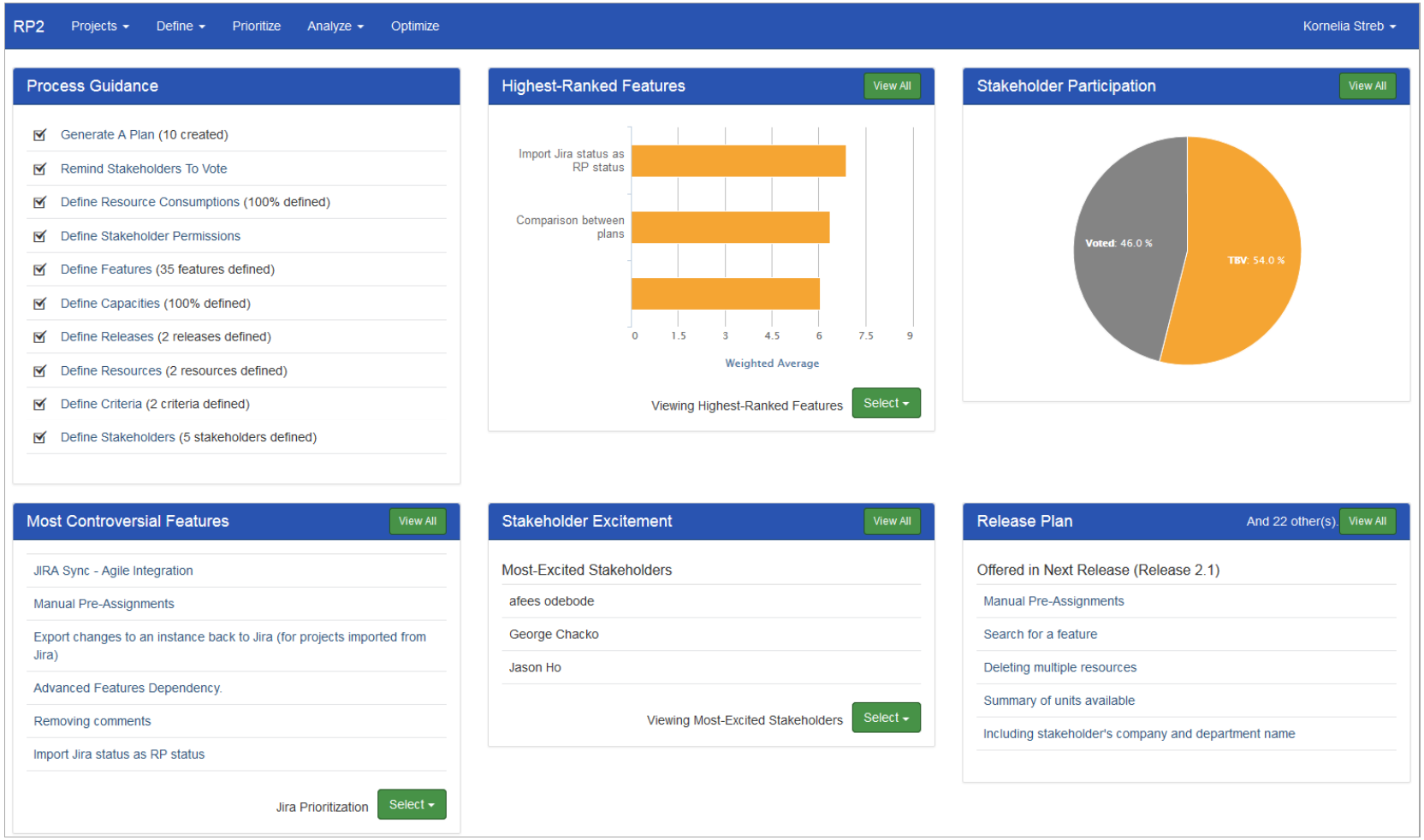

Figure 1: Dashboard of ReleasePlanner™ – a decision support system

In this thesis, we used ReleasePlanner™ [82] as a basis for further development in when-to-release and technical debt analysis and study. The ReleasePlanner™ system was chosen based on three different criteria, in relation to this research:

- Robustness and comprehensiveness: the tool has been used for numerous academic and industrial projects [5] with proven results in decision supports in release planning. The tool offers what-to-release planning with near-optimal alternatives and recommendations to product managers.
- Ease of access and ease of use: ReleasePlanner™ is web-based and can be accessed on-line with any web client. The tool provides interactive optimization, prioritization, and other release planning capability with powerful mathematical modeling, yet with simple and intuitive graphic interface, as illustrated in *Figure 1*.
- Availability of extension and code base for extension and plugin: There are a variety of plugins that are available and built into ReleasePlanner™. The W2RP plugin [48] provides interactive when-to-release decision support in consideration of features business value and release product quality. The API connectivity to issue monitoring tools, such as JIRA [36], are valuable in support for defects and testing types of technical debt. The plugins are further discussed, together with the new prototype implemented in this thesis as a new plugin, in Chapter Five:.

3. Overview of research problem

At the core of this research, we ask the following significant questions, that are motivated both from industry experience with organizations that utilizing software, and extensive related research work in the space of technical debt and when-to-release planning problem.

RQ1: How well do existing models, methods, or support tools efficiently and effectively model and manage technical debt?

RQ1.1: What are the current methods and metrics in measuring and quantifying technical debt?

RQ1.2: Do these metrics and methods identified above efficiently and effectively support the when-to-release decisions for (software) products?

RQ2: How can we evaluate, classify and formulate (both qualitatively and quantitatively) technical debt in relations to release decisions?

RQ2.1: How can we measure and estimate technical debt in relations to effort required to reduce it?

RQ2.2: How can we manage the impact of technical debt, in relations to when-to-release decisions?

RQ3: How can we provide decision support for creating and selecting the release plans that maximize readiness (business values and quality) while minimizing the negative impacts of technical debt?

RQ3.1: Is it feasible for release decisions to be supported by a prototype tool implemented based on the above proposed methodology?

RQ3.2: What are the characteristics required for the design of such prototype tool?

RQ3.3: How do we evaluate the methodology and validate the prototype tool in real-life, complex projects?

We will further examine the research questions as a formalized problem, and propose methodology and a prototype tool implementation to answer these questions.

4. Thesis structure

This thesis is organized in the following structure

- Chapter One: introduces the background of important concepts being considered in this research, coupled with the direction the research was conducted, presented as research questions (RQs).
- Chapter Two: examines related works that have been researched and the state of practice in industry and organizations, in the context of technical debt and release planning problems. This analysis of a rich pool of previously explored techniques provides us insights into the pros and cons of technical debt management, both in theory and practice, to create our model.
- Chapter Three: defines and formulates the when-to-release planning problem in the context of quantitative, predictive and manageable technical debt, release business values, and release quality. In doing so, we understand the different components involved and meaningful to the creation of a methodology and prototype tool that are later proposed in this thesis.
- Chapter Four: describes the methodology we proposed and designed as part of this research to address the research questions, both quantitatively and qualitatively, both theoretically and in application. The chapter explores in-depth how different types of technical debt can be measured, and managed, effectively in specific release scenarios, expressed as a combination of release time, business value, and reliability parameters.
- Chapter Five: outlines the architectural design, implementation, and application of a decision support system built based on the methodology proposed in Chapter 4.
- Chapter Six: evaluates the methodology and the prototype tool implementation. The evaluation is first done qualitatively through surveys, secondary literature

reviews, and retrospective data case collection. Secondly, the prototype tool is evaluated in different projects, from an interactive, scrum-based web application, to open source, frequently released projects. The findings and results of the evaluation are then analyzed to (i) appreciate the effectiveness of the methodology and the prototype tool, and (ii) address any potential threats to validity and correctness of the study.

- Chapter Seven: concludes the thesis, and offers future outlooks for researches that combine technical debt and release planning, in the same direction.

5. Contribution of this research

As illustrated in Chapter Two:, related work, the existing technical debt management models are qualitative, and technical debt remained invisible in the most part [14]. This research provides a concrete, quantitative, analytic-base, proactive methodology, that is capable of providing more accurate analyses of current debt portfolio in organizations, enabling better when-to-release decisions. The model has been tested by comparing analyses/predictions with empirically measured values in real-world technical projects and open source software. As such, the research in this thesis contributes to the current studies of Technical debt and Release planning in these main categories:

- Measurement and management of technical debt: providing abstraction rules, and an accompanied process guideline to systematically measure, monitor and manage technical debt in real world projects.
- A formalization and optimization of the when-to-release decision making problem (W2RP) in relation to technical debt through:
  - A what-if scenarios simulation approach to W2RP to enable analytic-based decision-making.

  - A multi-criteria optimization approach for trade-off solutions to release products with optimized portfolio of technical debt (minimizing the risk of negative debt and leveraging the opportunity of positive debt).

- A prototype tool implementation as a plugin to an existing release planning support tool:
  - To provide the support for decision-makers through interactive what-if data analysis,
  - To apply the methodology in real-world environment, and
  - To effectively observe and visualize technical debt decisions.
- Empirical evaluation of the methodology proposed and the tool implemented using case studies in different organizations, products, and projects profile.

Chapter Two: RELATED WORKS

1. Iterative software release planning

1.1. Release planning models and methods

There are many existing models and methods available for release planning, both strategic planning and operational planning, as systematically reviewed by Svanhberg et al. [93]. Intuitively, greedy algorithm is applied in some models, as a method of including as many high-valued features as possible in the next immediate releases. This method does not take into account costs, or quality concerns, or any of the trade-off measures that can potentially improve the values of the releases. On the other hand, Jung [53] modeled a cost-value trade-off analysis for the next release as a knapsack problem, maximizing the values of the features assigned, while assuring the total cost does not exceed the total capacity of a release. Similarly to greedy algorithm, as the model only consider cost and value, it is not possible to include other optimization criteria such as quality, technical debt, etc. and plan further than one release at a time.

Other models such as Bagnall et al.'s *Next Release Problem* [3] and Denne and Huang's *Incremental Funding Method* [28] improved the above methods by allowing some level of feature dependencies, while maintaining the greedy heuristic. However, resources constraints, which were essential to the when-to-release problem, are often overlooked by these methods. Other optimization criteria, as complex as composite technical debt, are also challenging to be included in these optimization models.

1.2. EVOLVE and EVOLVE II

In [41], Geer and Ruhe proposed EVOLVE method, which is an iterative solution method that utilizes the strength of genetic algorithms. In each iteration, a genetic algorithm is applied to determine the near optimal solutions, in relation to pre-defined objective

functions, based on assignment of features into releases. The method considers constraints and provides solutions for more than just the next releases.

Further to this methodology, in [82], Ruhe further developed EVOLVE into EVOLVE II, a comprehensive process, divided into 3 phases and 13 steps that employ the hybrid intelligence between advanced optimization algorithms and human experts' insights. The process and approach were implemented into ReleasePlanner™, which was also used as part of a pre-requisite in this thesis.

In this work, we apply the advanced process of release planning as proposed by EVOLVE II. We further enhance the approach to a bi-objective optimization, based on not only features function-points, but also technical debt, and release duration.

1.3. Multi-criteria search optimization

As analyzed in the previous sections, it is logical for the research to utilize multi-criteria search optimization. There are many points of consideration within and outside of technical debt concept. Internally, there are many types of technical debt that may have different impacts on the product release cycles at different point in time. Externally, product managers need to make decisions between balancing technical debts with release decisions, namely features offered, quality measures, and resource allocation. There is no single "best" solution, but a set of Pareto-optimal solutions in combination of all these aspects. Our approach identifies these solutions and offers them to human experts to make the final decision. Multi-criteria optimization, therefore, fits into this work's purpose.

There are a multitude of approaches and methodology in searched based Software Engineering, as reviewed and analyzed by M. Harman [45]. A fast search approached called NSGA II was proposed by in [26] to apply elitist offspring selection and clustering for better search results for multi-objective search optimization. T. Kremmel et al. [63] explores Software project portfolio optimization with advanced multi-objective

evolutionary algorithms, including NSGA II and other algorithms. The approach is validated with experimental case study to prove its ability to optimize based on multiple objectives to achieve highest strategic alignment value, while potentially identifying timeframes. As such, this thesis utilizes the advancements of these strategies, while applying our own membership functions and formulation of the project portfolio, technical debt, etc. to achieve optimality in generated trace-off solutions, which will be further modeled and explained in Chapter Three:.

2. Software release readiness – When-to-release Planning (W2RP)

2.1. Release readiness

Release readiness is often linked to reliability and testing activities on a project. In [72], Okumoto evaluated both the error detection rate and expected cost, in relation to release time. The authors determined the optimum time when testing can stop and the system is ready operational, with sensitivity to release time. In term of software reliability, there are multiple software error detection models and reliability growth models [40]. As scenarios are generated and evaluated based on what-if analysis, with a degree of uncertainty, readiness can only be evaluated predictively. Predictive models can be used to estimate software release readiness [78], in relation to defects and reliability. In [51], reliability growth model is notably evaluated based on testing-effort.

This research finds similarity with software reliability growth model in two ways: (i) the assumption that the more effort is invested into testing activities, the more defects will be discovered and fixed, resulting in a more reliable system and (ii) the amount of defects will eventually reach a finite number (potentially zero) when "enough" efforts have been invested into testing activities [100]. This is a valid model for the research, as defect detection and fix are also time-sensitive elements. Research in reliability such as [11] [29] indicated that defects that are discovered earlier are less costly, and vice versa. This was

parallel to technical debt, in a sense that postponed issue may not affect the system at the moment, but will grow in impact over time.

Different from strategic release planning, a project can only be released into operation once it is deemed "ready". As software development processes mature, release readiness needs to be evaluated both quantitatively based on metrics, often business related, and more qualitatively as well [2]. At any one point in time, project managers need to access the degree of readiness of a project or portfolio of project. Shahnewaz and Ruhe proposed an evaluation method using an index, called Release readiness index [90]. In this research, we formulate the when-to-release decision in a related procedure to the degree of readiness of a release, in consideration of technical debt.

2.2. Software estimation

In order to optimize the release date, effort estimation is an essential component, especially in software product development. Accurate estimation not only helps project teams to predictably plan for the work processes, but also prevents cost and schedule overrun [50]. There are many proposed methods for estimation and optimization. Ordered weighted average [101] is one such example of how aggregated operators can be applied to multi-criteria estimated from multiple sources. COCOMO II estimation model [9] is another empirical-based software estimation method that is widely used and recognized [43]. The model has three different phases. The Post-architectural model, which is developed after the general architecture of the software has been designed, takes into account both development and maintenance issues, with multiple scale factors and cost drivers [9]. The Post Architectural model is used as the inspiration to technical debt estimation in this thesis thanks to its ability to be further fine-tuned with empirical data, and its familiarity with participants from the software industry.

As software products and projects are extremely complex with a high degree of uncertainty, software estimation needs to be able to be calibrated and adapted to real

world projects. There are several improvements being studied and offered to calibrate the accuracy, and the ease of use of the method. Data mining techniques are used to leverage on previous data available in prior or legacy projects within the same product line or project teams [43]. Fuzzy logic was later introduced into COCOMO II to accommodate the high degree of variation and uncertainty in projects, especially projects that do not have a lot of prior data available for mining techniques. Huang et al. [50] have proposed a neuro-fuzzy logic to incorporate both fuzzy set and nuro-network into calibrating the cost drivers of COCOMO II in real life projects. These techniques all aimed at increasing the accuracy of the cost-drivers value; utilizing both mathematical simulation and available data, while maintaining the ease of use and collection of data.

In this research, we propose a technique that takes into account all these advanced in software estimation to estimate technical debt, using weighted-average of all the criteria in the dimensions of technical debt, and applying a user-centric model of effort adjustment factors, as outlined in details in Chapter Three:, Section 2 and Chapter Four:. The effort estimation for feature implementation and TD reduction will contribute towards how new features are being balanced against TD maintenance activity, given the time constraints of the release date.

2.3. When-to-release planning (W2RP) decisions

The decision to release the product sooner (with some degree of debt) or later (with accepted cost to opportunity for late release) can be formulated as a set of trade-off solutions. The product managers need to balance between the potential competitive advantage through faster delivery and the degree of readiness of the product (overall quality) and the added value through new and revised features. When-to-release decisions are largely re-actively using existing release planning tools such as IBM Focal Point[1], On-

[1] http://www-03.ibm.com/software/products/en/ratifocapoin

time[2] (for Scrum-based development) or ReleasePlanner [82]. At organizations with large system and established build and release engineering process, such as Facebook, when new features are in conflict with quality measure and release deadline, features have to be postponed in order to ensure quality [34]. However, in many organizations, the delivery of new features and change requests are often prioritized, typically at the compromise of quality and testing measure [35] [22].

The when-to-release problem is not often formulated as a trade-off solutions. McElroy and Ruhe [69] applied similar method with time-dependent variables for flexible release dates. However, the work focus on value of features across multiple releases, while not mentioning much about quality concerns, such as that of technical debt. Reversely, publications that focus on quality model and readiness of the release (as examined in section 1 above) do not consider time constraints as a major factor in releasing a product.

In this thesis, we proactively investigate the trade-off relationship between the total (possible) implemented functionality given the team capacity and the predicted quality achieved from the maintenance (TD reduction) effort investment. As a result, the product managers are empowered to see the projected impact of releasing earlier (or later) in terms of reduced (or added) functionality and/or quality. Similar approaches and tools support can be found from previous publications in the same group [48] [49], and from industry practices [27] [30].

3. Technical debt

3.1. Attributes and types of technical debt

There are multiple classifications of technical debt available. Many practitioners consider technical debt exclusively as a code quality measure [60] [62]. Publications in this category looks at technical debt only from the negative impact level, arguing that technical debt

[2] http://www.axosoft.com/

makes application harder to maintain, modify, and extend [80]. Others propose to view technical debt from a requirement engineering point of view, arguing that "technical debt in requirements is incurred when we decide prioritize requirements which are ultimately neither necessary nor deliver the most value to the customer" [33]. There are also multiple attributes of technical debt, from principle of debt, interest of debt, to visibility and alignment of debt [57]. Figure 2 illustrates the attributes and characteristics of TD that have been studied so far. The research presented in this thesis aims to combine both of these aspects to create a holistic view of technical debt in product and product portfolio in organizations.

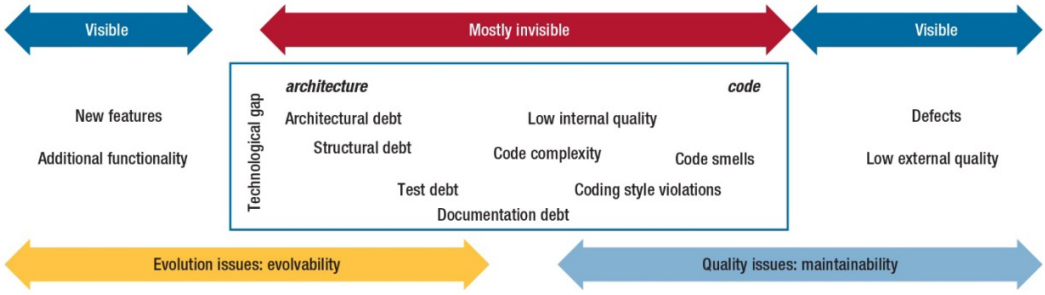

Figure 2: TD characteristics and attributes [57] [58]

The technical debt classification as proposed by N. Taksande [94] which includes: documentation, design, testing, and implementation debt, offers a more complete categorization. These types of technical debt cover both process-oriented debt (documentation and design), and code-related debt (testing and implementation). The classification also considers the benefits of technical debt, as opposed to just negative type of debt. The research, however, focuses on empirical studies of already incurred debt and the root cause of such historical debt data. In this research, we further this work by quantifying technical debt in relation to project specific artifacts (e.g. requirements specification, requirements traceability matrix, defect repository, etc.) and allow predictive analysis for strategic planning.

3.2. Current technical debt practices and methodology

In an industry survey of 35 practitioners, technical debt has been highlighted as an important challenge that needed to be addressed [62]. Recently, the alignment of requirements engineering and testing gained great interest in research and practice. Weak alignment of requirements engineering with testing may lead to problems in delivering the required products in time with the right quality [33]. However, there are very little publication or methodology that addressed the gap between requirements and actual deliverables as technical debt. Technical debts are present both in Agile practices and Lean practices in software development [76] [14] [59].

Furthermore, there have been an increasing trend in equating technical debt to the cost (in dollar or man power cost) to bring the system back to the optimal, desirable status, "cost to green" [32], or the opportunity cost to delays in schedules. This measurement was done via the remediation cost, based on a set of previously set requirements metrics [60] or through the available industry data to determine the cost to fix each instance of each type of technical dent [32]. Addressing the requirement gaps as part of technical debt, using quantifiable measures and metrics, both in term of dollar cost of opportunity, and man-power costs to bring current code to a more stable, desirable level of code will be both addressed and modelled in this thesis.

Lastly, methods such as SQALE used a remediation function to attach to every requirement that is identified as practices that will incur technical debt. The remediation function is formulated as time spent working on non-optimal code (interest to the project) or time required for refactoring and changing the code towards optimal (paying back debt principle) [60]. However, the authors only measure and maintain "internal debt that is associated to the source code", while ignoring other process oriented measure such as documentation, design, or even testing debt. In [58], it was argued that methodology that attach the dollar figures to compliance such as SQALE or CAIP

would potentially leave out systems that are "perfect" in term of debt, or having more estimated accumulated debt than operation cost or operational profit. This quantification and tracking of technical debt is rather re-active, based on the tracking in actual operation of the development and testing process. The SQALE method also based upon massive input data in each requirement levels that may not be available in real-time, making it hard to utilize the tool in "meaningful and practical way" [32]. We propose a more general estimation methodology that can utilize the benefits of quantification without a large effort in data collection and processing. The method proposed in this paper uses predictive analytics and analysis to recommend proactive planning decisions for the management of technical debt. We utilized similar approach in term of comparing the discrepancies and gap between functionality requirements and actual code implemented for our Testing and Defect Debt formulation.

3.3. Opportunity and risks management

Technical debt is often linked with risk [14]. If not managed effectively, over-accumulated technical debt can cause the application systems increasingly harder to maintain, change, and potentially affect business operations. However, companies and projects take on technical debt because the potential opportunities and benefits early delivery would bring. This is illustrated in Figure 2, adopted with modifications from [94]. Analyzing and understanding the duality of risk and opportunity of technical debt is crucial to managing it effectively. On the effect of opportunity and risk analysis and management, Boehm [10] explained the most common concepts, including leveraging factors and mitigation strategies. The work in this thesis, inspired by such general concepts, goes one step further in explicitly defining different factors in opportunity and risk as technical debt factors, therefore measuring and managing them as part of a greater portfolio management approach. More details of this model can be found in Section III.B.

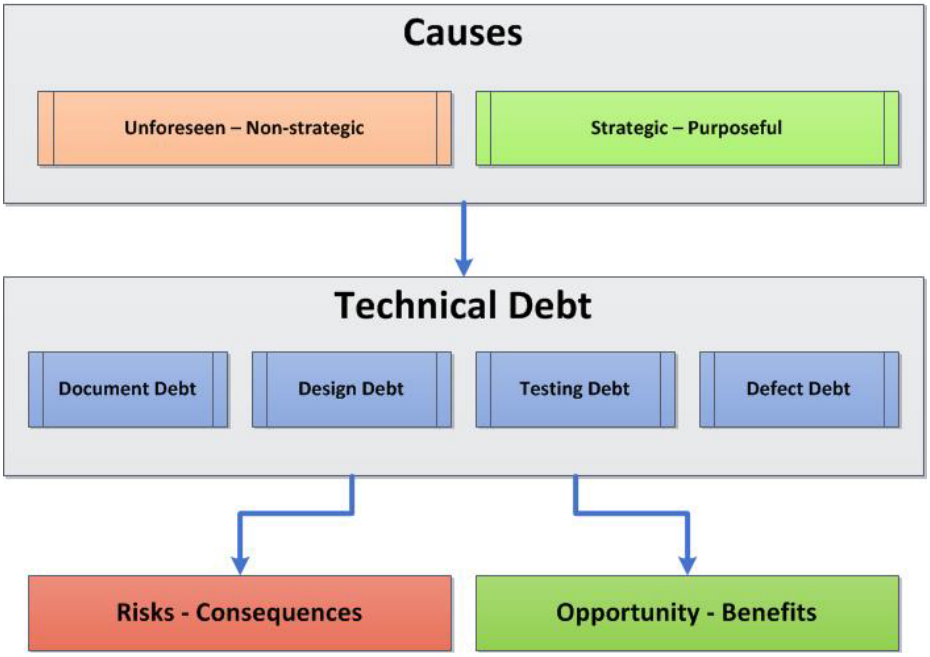

Figure 3: Technical debt evaluation model

Furthermore, as software products and projects are inherently complex, with a large degree of uncertainty, risks and opportunities are often not deterministic and time-sensitive. Technical debt is hard to measure quantitatively and is often overlooked by business users due to its uncertainty and lack of visibility. In [18], M. Cantor proposed an approach to calculate return on investment (ROI) of projects portfolio based on Monte-Carlos simulation [81]. In this research, in a similar manner, utilizing the results of Monte Carlos simulation, we identify a time-sensitive value to releases and projects, given its corresponding debt portfolio, balancing between risk and opportunity, as a part of our consideration for trade-off solutions generation and selection.

Lastly, the concept of release time determination based on risk evaluation with consideration to costs of delays has also been explored previously. X. Li [61] formulated the costs of delays, the probability of such risk when such delays happen, with sensitivity to release time of projects. Such formulation is similar to our concepts of what-if playing scenarios, further explored in Chapter Four:.

Chapter Three: PROBLEM DEFINITION AND FORMULATION OF KEY CONCEPTS

In this chapter, we define, and formulate mathematically where appropriate, the different key concepts that impact when-to-release decisions and technical debt decisions, independently and in relation to each other. Lastly, we formulate the W2RP-TD decision as a bi-objective optimization problem, so that it can be simulated, optimized and validated in the methodology and prototype tool implementation.

The definition and formulation of the problem is the first of its kind, at the point of writing, for Technical debt generally and Technical debt in When-to-release context specifically. The components of technical debt, and the proposed estimation methods associated with such components, are empirical based on a number of different studies. These studies are detailed in Chapter Six:.

1. W2RP formulation

1.1. Release decisions and variables

Decisions in product release planning are primarily related to features and their assignment to releases. Consequently, decision variables are related to a set of features under consideration. In this chapter, we explore the modeling of releases and features, following the well-established researches in this field by Ruhe [82].

Definition 1: Let F = {f(1),…,f(N)} be a set of N candidate features. At this point, the assumption is that the features are given. Their (given) description is used to prioritize them.

For effective planning, stakeholders and project teams need to decide how many releases they want to plan for. The model in [82] goes beyond the next release, exploring up to K releases of the product. In the case of trade-off solutions planning, as studied

specifically in this thesis, we only consider which features are assigned to the next release (release K=1) and the impact of postponed features (release K+1).

Definition 2: The decision variables x(n) (n = 1…N) describing the proposed strategy for handling features f(n). Each individual release plan describes the strategy for all the candidate features and is characterized by a vector x = (x(1),x(2),…,x(N)) with

x(n) = K if feature f(n) is offered at release K (n = 1…N)

x(n) = K+1 if feature f(n) is postponed, i.e., it is not offered in one of the next K product releases (n = 1…N)

In the case of planning when-to-release in consideration of technical debt, we only consider K=1. Therefore, the decisions support simply means x(n) = 1 or x(n) is postponed.

Assumption 1: In this thesis, and in general in software-driven decision making process, we assume that feature are largely independent, or can be formulated as such. In the event of interdependencies, these independencies can be formulated and solved within the optimization. The exploration of advanced constraints in release planning decisions are explained and solved in [77].

1.2. Resource consumption vs. capability of resource

In order to implement a new feature, or fix a defect, resources are required. The most common forms of resources are human resource and monetary costs (such as machine cost, etc.). In our studies, the cost of implementing new features, or fixing existing defects, can be traded off with effort needed to reduce technical, to improve the overall satisfaction of the project. We denote the effort required to build a feature f(n) as E(n). Resources in real life projects are not unlimited. There is a limit to the total resources available in each release of the product, called the capacity of that release.

Definition 3: For the next release k = 1, the capacity is denoted by Cap. More formally, as defined in [82], a feasible release plan needs to satisfy the capacity constraint:

$$\sum_{n:x(n)=1} E(n) \leq Cap \quad \text{EQ. 1}$$

With x(n) being either new features, change requests, or defect fixes.

1.3. Business value of a release

When selecting features to be assigned into the next release (or a general release k), product managers have to evaluate and prioritize the potential values of such features. The total value will potentially contribute to the final release. This business value can be determined based on a 9-point scale from 'Very Low' value to 'Very High' value [82]. In real-world project, business value is often evaluated based on the potential revenue (in dollars) that the offering of the release or features will bring. This can also be easily translated or mapped into the 9-point scale as mentioned above, and as illustrated in the case study in [35]. In this research, we denote the business value of a feature f(n) as V(n). Each release offers a set of features.

Definition 4: For the release next release k = 1, the total release business value is denoted by TRV. The total business value of a release is the sum of values of all features being offered in that release:

$$TRV = \sum_{n:x(n)=1} V(n) \quad \text{EQ. 2}$$

Our goal in this optimization is to maximize the value of TRV the next release, given the available resources assigned to the release.

2. Technical debt formulation

2.1. Goals, Questions and Metrics (GQM) in defining technical debt

Based on literature review, there are many available and established classifications to evaluate and track technical debt quantitatively. In this research, we formulate technical debt as the (units of) effort required to refactor and repay debt, or "time to green", as proposed by R.J. Eisenberg [32]. Similar to SQALE methodology [60], remedial functions

and efforts associated with them are defined upfront. However, instead of fine granularity for each functionality or metric, we use a nine-point-scale evaluation. Based on related work in technical debt dimensions, we use a set of metrics that are defined applying the GQM (Goal-Questions-Metrics) framework [4].

There are benefits to applying GQM for measuring of TD: (i) there is a level of consistency in measurement of technical debt and the gap between actual code and desired level of code, as defined up front in specific organization; (ii) the cost of evaluating and management of technical debt will be reduced, as we do not need human experts to repeatedly define new remedial functions or estimates; (iii) project team can replicate and fine-tune their evaluation from legacy projects to future projects. Below we propose the dimensions of technical debt to be evaluated, coupled with concrete examples that will later on be illustrated during our case studies.

Compared to other methods existing from literature (Chapter Two:) and from industry practice (Chapter Six: case study 2 and case study 3), GQM allowed project teams to be generic enough from early stage of the project to define what TD means to them, yet enabled fine-tuning and accurate metrics-driven analysis at the later stage.

An example of dimensions and criteria of TD is listed in Table *1* below.

Table 1: Dimensions for technical debt metrics

Dimension	**Criteria**	**Examples**
Process rules compliance	Documentation completeness	The degree of not-available documentation in artefacts or delivered code.
	Availability of Interface (API)	The percentage of non-documented interfaces.

Dimension	Criteria	Examples
Quality testing	Reliability	Remaining defects after release, total availability.
	Code Coverage	The percentage of code that is not tested, or not covered by automated tests
	Security	The potential threat to security due to outdated, broken code
Maintainability	Components coupling	The interdependency between components
	Reusability	The percentage of duplicate code
	Defect detection rate	The frequency of new defects or issues being detected
Complexity	Complexity	Size, KLOC, degree of legacy system available

In order to measure these dimensions of technical debt, we proposed a set of questions and metrics (or criteria) to evaluate them. The following table is just an illustrative example of how GQM is being applied. In reality, this guideline can be different from project to project, depending on the nature of the product, and the organizational practices.

Table 2: Example of the Goals, Questions, and Metrics (GQM) to measure TD

Goals	Questions	Metrics / Criteria

Goals	**Questions**	**Metrics / Criteria**
Assess process rules compliance debt	How complete is your documentation, at the code level and formalized level?	Available documentation in: - Delivered code - Artefacts
	How available are your interfaces and API, internally and externally?	Available API document: - Internally - Externally
Quality testing	How reliable is the software after released?	- Number of defects after release - Availability
	How much code coverage does testing provide on the code base?	- Percentage of code that is not tested - Percentage of code not covered by automated tests
	How secure is the software after release?	Number of identified potential threats
Maintainability	How many components are tightly coupled?	Number of dependencies between components
	How much code can be reused?	Percentage of duplicated code

Goals	Questions	Metrics / Criteria
Complexity	How large and complex is the system?	- Code size (KLOC) - Degree of legacy

It is important for product teams to define these Goals, Questions, and Metrics upfront as they will guide the general direction of technical debt management effort, as well as indicating the right prioritization in selection of Trade-off solutions. Our final goal is to evaluate the overall accumulated technical debt, relatively to the effort required to potentially pay off all these debt, through the four different dimensions listed above. For each dimension, the questions are designed to explicitly explore the different criteria. Furthermore, for each criterion c_i, a membership function $\mu c_i(F_k)$ is defined for a release k of feature set F_N. The value is describing the status of the current features set F_k, in terms of the degree of technical debt related to this criterion for the release k. The (normalized) value of these membership functions is given by the human experts. It can be partially experience based from past legacy projects, and partially based on objective metrics.

Definition 5: Let $C_d = \{c_1, c_2, \dots c_d\}$ be the set of criteria in determining technical debt for a TD dimension d. For each criteria c_i in dimension d, we define $\mu c_i(F_k)$ as the degree of technical debt in that criteria for the release k, $\mu c_i(F_k) \in [0,1]$. The value of each membership function is determined based on previous project data or stakeholders' satisfaction. Furthermore, the human experts of the project assign a (normalized) relative importance weight w_{d_i} so that:

$$\sum_{i=1}^{4} w_{d_i} = 1 \qquad \text{EQ. 3}$$

In this research, membership function is not explicitly defined as a mean for different project teams to define their own technical debt profile: with 0 means there is no technical debt in that criterion, and 1 means the technical debt in that criterion has

reached its upper threshold and needed to be eliminate before new functionality can be implemented. In our case study 1 and case study 2 in Chapter Six:, different open source system (OSS) projects and a proprietary project defined TD in a context-specific membership score based on the satisfaction of stakeholders in each category. However, having the dimensions and criteria clearly defined, coupled with the evaluation process, have helped teams identify and measure their TD profile effectively.

2.2. Estimation of effort for technical debt reduction

Based on the above dimensions, we propose a method to estimate technical debt. Intuitively, as the project grows in size and complexity (measured by KLOC), the size of technical debt and the effort to address it will also grow proportionally. We will validate this hypothesis with a case study and experiments in real world data. Here, we use an exponential measure, inspired by the COCOMO II [9] technique. In this measurement, we assume that the size of the project is known or previously estimated. The measure here is most similar to Post Architecture phase in COCOMO because TD incurred and accumulated during the implementation itself, and not in the composite phase.

In this context, technical debt is measured by total work-effort (in percentage) required to completely eliminate the accumulated debt within the project. Relative percentage is used to account for systems that may appear as "having more estimated accumulated debt than operation cost or operational profit" [58]. Furthermore, industry case study reveals that, while practitioners are generally aware of the impact of technical debt, they are often constrained by how much effort can be invested in activities that reduce debt [62] [46].

Similar to COCOMO II method of estimation, the product team first evaluates the code size and complexity (represented in KLOC). Given the specific context of each project, this code complexity, and hence its technical debt, will scale in term of effort required to implement. As technical debt is evaluated on the same project, effort

adjustment factors (EAF), such as platform, precedence, team proficiency, etc., will be the same for TD and implementation.

Definition 6: Let E_{TD} be the effort required to eliminate accumulated technical debt in the current release. The percentage of accumulated technical debt in the project, called TTD, is estimated by:

$$\text{TTD} = \frac{E_{TD}}{E_K} = \frac{(KLOC)^{TDAF} * EAF}{(KLOC)^{SF} * EAF} = \frac{(KLOC)^{TDAF}}{(KLOC)^{SF}} \quad \text{EQ. 4}$$

where KLOC is the size of the project, as measured by thousand lines of codes

SF is the scale factors, as defined by COCOMO II [9]

TDAF is the technical debt adjustment factor

E_{TD} is the estimated total effort required (in person-month) to eliminate TD

E_K is the estimated total effort required (in person-month) to complete release K

Definition 7: The technical debt adjustment factor (TDAF) is the weighted average (WA) of all criteria, as specified previously, in all 4 dimensions of technical debt. Weighted average is calculated based on the score of all criteria, as an aggregation operator to consolidate the adjustment factors available. This value is used to collectively combine the different evaluation of technical debt in different criteria, and dimensions.

$$\text{TDAF} = WA(\sum_{i=1}^{4} w_i d_i) = WA\ (\sum_{i=1}^{4} \sum_{j=1}^{di} w_{ij} \mu_{c_j}) \quad \text{EQ. 5}$$

Semantically, the larger the value of TDAF, the higher the growth rate of effort required to pay off technical debt. As the data for technical debt measurement can be uncertain, in these criteria, fuzzy set theory will be beneficial to manage uncertainty in cost estimation, reliability prediction and requirement analysis. In this context, each technical debt criteria takes on a "membership function". The application of the fuzzy approach in the context of COCOMO II is furthered explored in [66].

Assumption 2: In this research, we assume that the estimated effort required in completing the next release (across all activity streams such as design, implementation, testing, etc.) is known. This is a reasonable assumption as software estimation practice is a well-established field since at least the 1960s [9]. The COCOMO II approach, similar to this estimation of technical debt, has been validated in multiple software projects across different industries [43].

To observe this estimation process in practice, please refer to section VI.B for a case studies and Appendix A for example calculations.

3. Problem formulation: W2RP-TD trade-off solutions as bi-objective optimization

Definition 8: The *release planning problem* RPP consists of finding an operational plan and a feature set F_0 to be implemented. Therein,

- F_0 is a subset of the given feature set F;
- A set of technological constraints TC have to be satisfied for implementation and quality assurance;
- A pool of developers with individual capabilities is available to perform the implementation and quality assurance tasks, and
- A fixed release date RD is given for having done all the implementation and testing of features from F_0.

Release engineering, among others, encompasses the question to find the most appropriate release date in consideration of release readiness and business value. In order to make this decision, different scenarios for release time need to be evaluated pro-actively. Based on the initial (baseline) when-to-release plan (RD_0, F_0) characterized by the original release date RD_0 and an initial feature set F_0, we study a series of release scenarios from varying the different release parameters. Each of these scenarios is a

variation of the original release plan, and is determined from (RD_0, F_0) by re-balancing the effort allocated to functionality implementation and refactoring technical debt, so that the total effort required for that release is still within the capacity constraint of that release.

The objectives in this optimization and selection are (i) shortening the release cycle date as much as possible, and (ii) maximizing the value of features being offered as part of the release and (iii) minimizing the negative impact of technical debt; while maintaining the capacity resource constraints of the original release plan.

As analyzed in background and related works, there are both opportunities and risks that are attached to technical debt. However, as the nature of impact of debt is prognostic, there is a degree of uncertainty involved in the evaluation and reduction of technical debt. Furthermore, in organizations, the argument for not addressing technical debt is that it is not measureable. The impact needs to be attached to a tangible, financial figure to be meaningful [62]. Lastly, technical debt changes over time. An existing issue can potentially be indirectly fixed by a refactoring in the code base, for instance. Taking into consideration of all the degrees of uncertainty, this research attempts to formulate future risk and opportunity in technical debt using Monte Carlos simulation [81] [18] and decision tree risk management [7]. The aim of this simulation is to come up with an estimated value to influence trade-off decisions, yet remain easy enough to apply in real-life operations.

Inspired by the decision tree approach for risk management proposed by B. Boehm [7], we modeled technical debt at a point of time t as a profile of a combination of decisions being made about which features to be included, which to be left out, and to which degree does accumulated TD is being reduced. At each point t, this profile of features being offered, coupled with the potential value and the total cost of implementing them, is traded off against the accumulated TD in that profile.

Definition 9: Pareto optimization is the process of finding a set of non-dominated solutions from a pool of candidate solutions. For the case of multiple objectives $F_1, \ldots F_k$, a Pareto-optimal solution, also called trade-off is one where there is no other solution being better in a vector dominance relation defined by $(F_1, \ldots F_k)$ [86].

Definition 10: For a given sequence (scenario's) of release dates RD_i (i = 1 …L), the when-to-release problem W2RP means to solve a sequence $\{RPP_i\}$ of problems RPP, each of them with a corresponding release date RD_i. We consider a baseline release plan F_0 and its vector $(TRV(F_0), TD(F_0), RD(F_0))$ related to value, technical debt respectively release date. Then W2RP means to

- Determine operational release plans based on varying feature sets F_i. which are
- Pareto-optimal solutions among all the variations of possible plans in terms of the two criteria
  - F_1 = Maximize $\Delta TRV(F_i)$ (with $\Delta TRV(F_i) = TRV(F_i) - TRV(F_0)$)
  - F_2 = Maximize $\Delta TD(F_i)$ (with $\Delta TD(F_i) = TD(F_0) - TD(F_i)$)

Therefore, W2RP problem is a sequence of bi-objective optimization problems, each for a fixed release date and with two objectives: total release business values (TRV) and total accumulated technical debt (TD). As resources can be transitioned between activities of implementing new features and maintaining (reducing) technical debt of other features, and is ultimately capped by total capacity of the release, this problem can be solved by optimizing resource allocation [47].

Assumption 3: Effort can be transferred and trade-off seamlessly between implementation activity and maintenance activity.

This is a rational assumption in project management practices [10], given that the resources assigned to fix defects or TD are the resources who implemented these features. This assumption simplifies the methodology and estimate used in later sessions. In some

complex projects, there may-be overhead in task-switching for developers. It needs further fine-tuning in this case, which is enabled by the COCOMO-inspired adjustment factors.

Chapter Four: METHODOLOGY

We have established in previous chapters that the decisions of when-to-release a product is a complex, multi-faceted process. In consideration of technical debt (TD), which has embedded implication of predictive risk-opportunity analysis, the assessment becomes inherently more challenging. In Chapter Six:, case study 1 highlighted the existing TD in multiple open source software projects, and its potential impact on the long term stability and development of the products. Furthermore, in case study 2, the TD profile is defined and monitored differently from what is commonly known from literature. Therefore, in our methodology, we aim to combine the power of tool-support data monitoring with the domain experts' opinions in explorative scenario simulation and selection. This methodology does not advocate replacing product managers; it promotes informed decision-making process by providing (all computational) possible alternatives of release plans, with visible and detailed profile of release date, features, and technical debt in each alternative.

The methodology allows organizations to first define their own TD profile, based on what have been suggested by literature reviews and industry best practices, using GQM framework. We then formulate the problem by continually monitoring existing legacy data in these defined categories. During planning process, product managers explore different scenarios given their current TD profile, features to be implemented, and the desired release dates of the next release. Lastly, but most importantly, this process can be repeated several times, and on demand, as the current situations of product constantly change. The tool keeps historical data of past plans and accommodates comparisons with any new 'what-if' scenarios. This workflow is the outcome of empirical studies (Chapter Six:) and explorative strategic planning with project managers from Smart Inc. (case study 2) and Expert Decisions Inc. (case study 3).

We propose a methodology to systematically solve the W2RP-TD problem, as defined in Chapter Three:, using the following workflow:

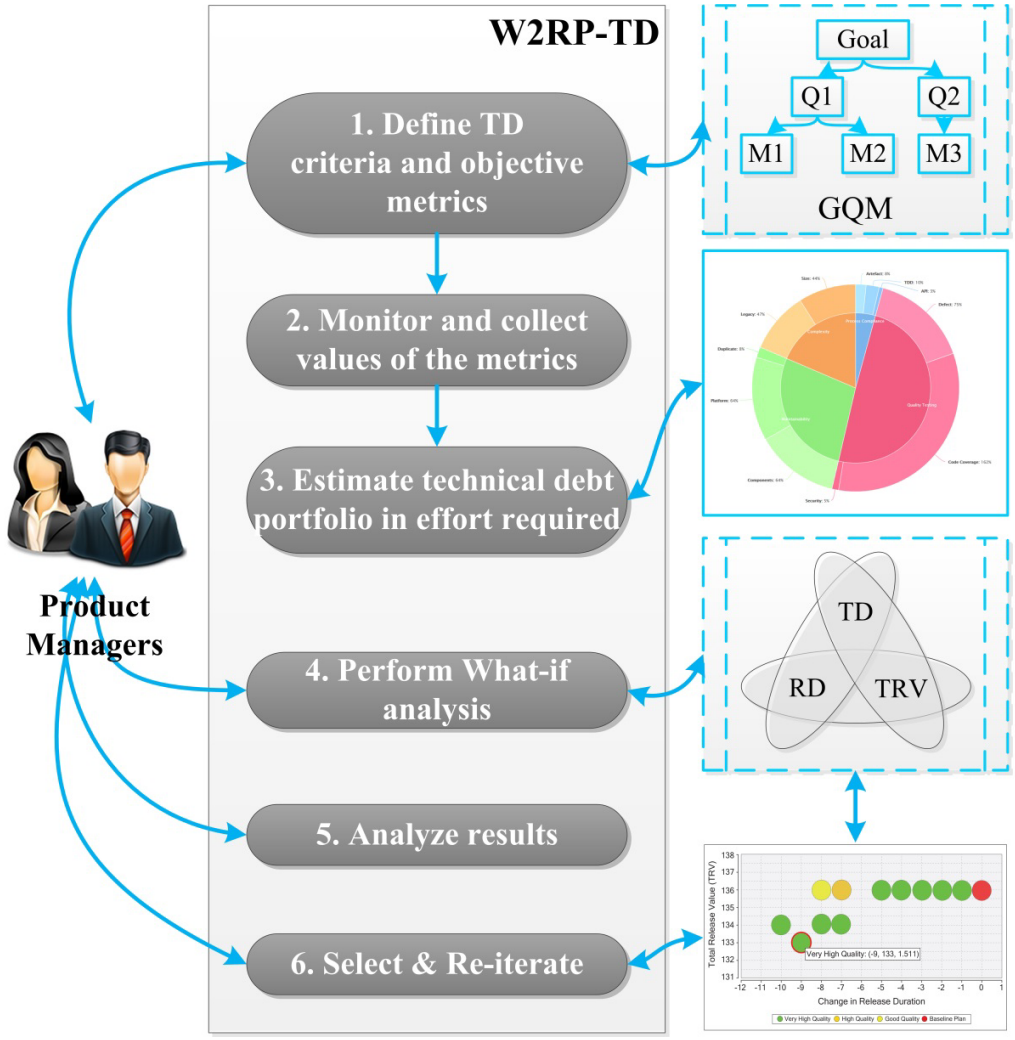

Figure 4: The W2RP-TD methodology

1. Proposed Workflow

When-to-release planning in consideration of technical debt (W2RP-TD) follows an explorative strategy. The technique generates explorative solutions within a set of original constraint parameters:

- ΔRD is the maximum number of days the release duration can be varied (directly related to capacity $Cap(F_i)$),
- $\Delta TRV(F_i)$ is the change in value due to (number of) functionality change
- $\Delta TD(F_i)$ is the percentage of remaining (accumulated) technical debt in that release.

Users can configure the feasible range of changes for all three dimensions, based on historical data or project specific needs. These factors control the degree of acceptable change in release duration (RD), value (V) and technical debt (TD). The problem is a bi-objective optimization (by varying one of the three factors, we maximize the combination of the remaining two factors). As described in the pseudo-code (Section 2), we follow NSGA-II, a fast and elitist multi-objective genetic algorithm for this optimization problem [26]. NSGA-II is used because of the algorithm's ability to search for Pareto-optimal solutions (*Definition 9*) while account for potential constraints such as capacity and dependency. Efforts from refactoring (reducing TD) or implementation (increasing V) are exchanged, up to the threshold of (pre-set) reduction factors, and new solutions are generated. The W2RP-TD technique is to determine as set of trade-off release plans generated from a series scenarios defined by the user from this exchange.

More specifically, the steps of execution in this explorative method, with reference to the diagram in Figure 4, are as followed:

Step 1: *Define TD criteria and metrics*: allow managers to set project-specific questions and metrics that are relevant to the goal of tracking and monitoring debt, together with their relative weight of importance.

Step 2: *Monitor and collect values*: follow the project during operation, before release date, to collect the data to measure technical debt.

Step 3: *Estimate technical debt* for the initial release date RD_0: At any point in time, with the data collected, effort to address technical debt can be estimated by EQ. 4.

Step 4: *Perform what-if analysis*: user can interactively view the implications of varying the release duration, refactoring TD or modifying the set of features offered.

Step 5: *Analyze results*: from all the scenarios explored, a pool of release plans is generated. Analyze results eliminates all plans that are dominated by other(s). All trade-off solutions will be maintained as candidate release alternatives.

Step 6: In *Select and re-iterate*, the product manager(s) can either go back to define another scenario or terminate the process and select the most attractive plan, representing the best balance between the criteria.

At the end of step 6, product managers are offered a set of alternatives of feasible and highly optimized solutions based on the parameters they have previously defined. At this point, they are empowered to make informed decisions based on the alternatives available. Furthermore, with this procedure, they can explore multiple sets of (what-if) parameters, and repeat the process as often as desired until the most attractive plan(s) are found.

2. Bi-objective optimization for W2RP-TD

In this section, we further discuss the algorithm that is being used in Steps 4 and 5 of the above proposed procedure. As the problem is formulated as a bi-objective optimization, the solution is inspired by existing evolutionary search based techniques, such as EA, NGSA-II, etc. [45] [95]. This methodology is not about re-inventing search base software engineering, but using existing known techniques appropriately to support the decisions modeled under W2RP-TD in Chapter Three:.

In this optimization, a plan F_i is comprised of 3 elements, release duration (RD), business values (TRV) and accumulated technical debt (TD). The values of each element are inter-related to the resources and capacity of the projects and releases. Release duration is defined by calendar (or business) dates, which in turn determines how many person-days (or similar measures such as person-months, etc.) is available for each plan. The

business values will be a real (positive) number of the mapping of combined features business score. TD is the most complex measures, which is a real number between [0…1] of how much TD is accumulated in a project at the point of release.

Each plan is constrained by the total capacity available in the next release *Cap* (EQ.1). A plan is generated by assigning a diversified set of features into the release, so that the above constraint is satisfied. The generation of plan is described in details by Ruhe in [82]. There are multiple approaches available for this generation, such as Knapsack, Greedy, and EVOLVE. From the pool of generated solutions, we optimized by searching for Pareto-optimal solutions, defined by *Definition 9*. In each iteration, the solutions are re-iterated by operations to generate new release plans. The optimization will halt when there is no possible plan can be generated, or the time limit of execution (as specified by users) has been reached.

The profile of diversified solutions in the non-dominated solution space will be presented to users, in graphical representation, to facilitate the decision making process.

The pseudo-code of this approach is described in the following table:

Table 3: Pseudo-code for W2RP-TD search base algorithm

Function Generate Solutions (F_0:=Baseline plan, F_N:= Features pool, ΔRD:=change in release duration)

Define: S := Solution pool, S* := Trade-off solution pool

Define Baseline $F_0 \leftarrow (TRV(F_0), TD(F_0), RD(F_0))$

Initialize solution pool S ← [ ]

While not TerminateCondition() do

Vary RDi: $\Delta RD \leq Max(\Delta RD)$

For each RDi, generate plans F_i so that $E(F_i) = \sum E(f(i)) + \sum E_{TD}(f(j)) \leq Cap(RD_i)$

For all features f(1,…,N), add f(k) into F_i (using methods described in [82])

$E(F_i) = E(f(k)) + E_{TD}(f(k))$

```
            If constraint E(Fi) ≤ Cap(RDi) is met
                NewPlan Fi ← Fi (TRV(Fi), TD(Fi), RDi)
        For all F* in S*:
            If Fi is superior to F*: Eliminate F* from S*
            If F* is superior to Fi: Eliminate Fi from S
    Merge (S, S*)
Return trade-off solutions S*
```

For illustrative (numerical) examples, please refer to Appendix A.

The results of this approach are implemented in Chapter Five:, with empirical data collection and analysis to be presented in Chapter Six: through three different case studies.

3. Evaluation of the bi-objective optimization approach

In this research, we used bi-objective optimization techniques as the approach to select the trade-off solutions for decision-making. It is noteworthy that there are many possible optimization approaches and techniques to achieve the same purpose, each with their strength and weaknesses, as analyzed in [45]. We selected bi-objective, as a scale-down version of the full multi-objective optimization as the criteria in the W2RP-TD problem space is well defined between RD, TRV, and TTD. Furthermore, the approach enables for accommodation, at some levels, constraint and dependency handling [95].

In our case studies conducted as part of this research (Chapter Six:), bi-objective optimization was able to quickly generate solution in relatively short period of wait time (less than 30 seconds in most cases) while maintaining the degree of diversity and optimality of the release plans, as evaluated by experts on the project team. The method is still relatively novel, with only three case studies so far, so more evaluation is needed to identify the best optimization approach for the W2RP-TD problem.

Chapter Five: PLUGIN IMPLEMENTATION

1. Motivations

As analyzed in the previous chapters, technical debt is best managed when it is visible, and quantifiable. Survey from industry participants [62] has indicated that managers want a way to manage technical debt and communicate its consequences in organizations. They need a user-friendly and business-oriented tool to achieve this.

Furthermore, in the technical debt (TD) definition and metrics evaluation, we find resonance between the technical debt metaphor and the release planning philosophy. In [73], the author argues that, software aging is inevitable and must be designed for change early in the process. Software engineers not only have to change their mindset about not just getting the code works the first time, but also maintaining its changeability in the future, by applying a combination of best practices in documentation and coding. Release planning for more than one release ahead is where technical debt management will make the most impact if it is monitored, reviewed, and managed frequently.

Currently, in TD management and tracking, there are several existing plugins and web interface to estimate principle of TD (but not the interest of TD). Specifically, there are tools such as the SonarQube™ Plugin, implementing SQALE [38], CAST's method of technical debt estimation [25], and Marinescu's method of technical debt estimation using design disharmonies [67]. However, the authors only measure and maintain "internal debt that is associated to the source code", while ignoring other process oriented measure such as documentation, design, or even testing debt. In [58], it is argued that methodology that attach the dollar figures to TD will potentially leave out systems that have more estimated accumulated debt than operation cost or operational profit. Furthermore, in [42], the authors found little similarity and correlation between these estimates with the business value and effort invested in features.

Lastly, inspired by Dialog Generation and Management (Decision Support) System, as defined by A.P. Sage [85], the prototype solution in this thesis is designed to be interactive and graphical oriented. With the development of current web applications and graphic user interface (GUI) [64], the tool is designed using Model-View-Control (MVC) model. The detailed design of each component will be described in section B of this chapter. In the context of when-to-release, there is no specific tool or plugin that handle release date specifically. The closest set of tools are project management type of tools such as Microsoft Projects, JIRA, OnTime, etc. However, these tools focus on re-active aspect of release planning. Given the current development velocity, and issues being reported and tracked, they envisage if the projects will be released on time (based on a pre-determined date). The only tool that potentially incorporate prediction of when-to-release decisions and its impact on risk and opportunity is the ANDES tool by IBM[3], which is not publicly available at the point of writing.

The W2RP-TD plugin tool we designed take into account these short-coming in both dimensions, utilizing both strategic planning capability (from ReleasePlanner™) and the operational planning capability (through issues tracking and TD estimation).

The prototype tool is designed as a plugin into an existing web application ReleasePlanner™ [82]. The motivation of this design decision is to leverage on existing strengths of ReleasePlanner™ in the following capacity:

- Generate near-optimal what-to-release plans based on a set of (well-defined) features, criteria, and priority.
- Integrate with different plugins, including issue tracking plugin (JIRA) [36], and when-to-release planning (W2RP) that were previously developed.

[3] http://researcher.watson.ibm.com/researcher/files/us-duester/WhitePaperANDES.pdf

- Monitor and display dashboards of current status of release plans and features implementation.

Furthermore, the plugin extends these capabilities in the following directions:

- Monitor and represent technical debt graphically and interactively during the project development life cycle via added widget in the dashboard.
- Allow direct manipulation of what-if solutions, where users can interactively manipulate the release date of a product or project, and see the predictive results of such changes in real-time.
- Allow archiving and comparing of solutions generated at different points of time.

The design is an extension from previous published work in the same direction of W2RP implementation [48]. The below blue-print and screenshots only describe the extension with technical debt plug-in.

2. Solution blue-print design

We consider the major classes of the solution in the following (simplified) Unified Modeling Language (UML) class diagram.

We start by designing two objects fundamental to release planning, i.e. *Project* and *Feature*. In ReleasePlanner™, the objects have been designed and used as part of planning. The plug-in objects extend these existing objects with new fields that are specific to technical debt (TD) and release date (RD). For features, existing estimates such as implementation effort and testing effort are used directly from ReleasePlanner.

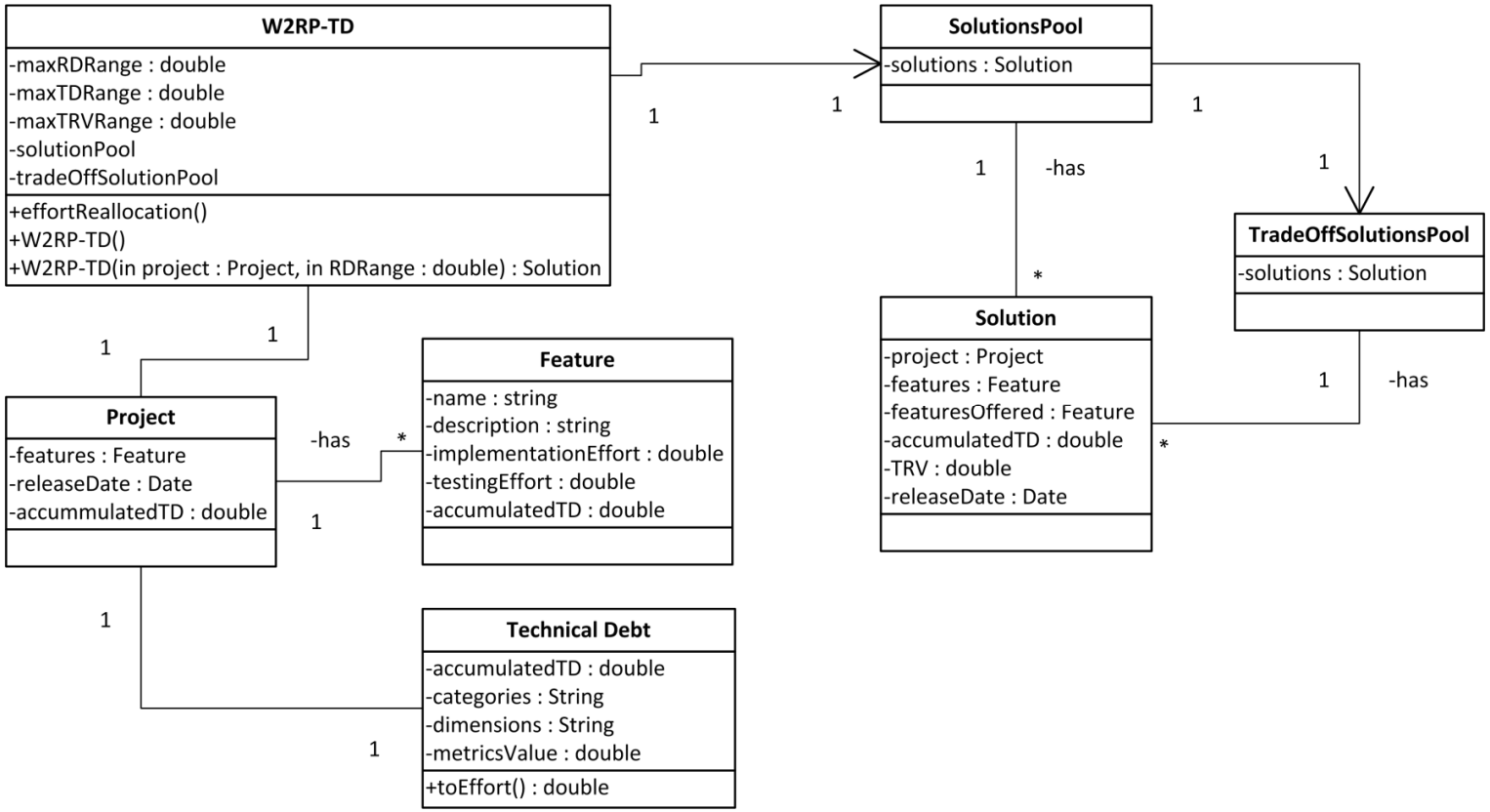

Figure 5: UML model of the W2RP-TD Plugin Solution Design

The *TechnicalDebt* object is a new object designed to track and estimate technical debt (TD). It is a composite of all TD categories. The object stores all the TD dimensions and criteria in array of Strings. The accumulated TD is the composite measure from all the defined TD categories (represented in TDAF). The *toEffort*() method returns the estimated effort required to address existing accumulated TD, applying *Definition 6* , EQ 4:

$$\text{TD} = \frac{E_{TD}}{E_k} = \frac{(KLOC)^{TDAF}}{(KLOC)^{SF}}$$

The *Solution* object is design to hold all feasible solutions derived from the project. The solution is generated by ReleasePlanner™ tool by allocating features into the next release within the constraints of capacity and resources, determined by the desired original release date. The algorithm and engine behind this solution generation is described in Chapter Four:, Section 2 [82]. Each solution will have the list of features being implemented, the total release value (TRV) and release date (RD) associated with them. All feasible solutions are stored in the *SolutionPool* object.

Then, the W2RP-TD engine will compare all possible solutions in the solution pool to identify non-dominated (trade-off) solutions. These solutions are then compared with the existing trade-off solutions in the *TradeOffSolutionPool* object. If the new solution is being dominated by existing trade-off solution, it is simply removed from *SolutionPool*. If the new solution dominates a solution in trade-off pool: that dominated solution is removed, and the new solution is added. If no dominated solution is found, the new solution is added into trade-off pool.

After this operation, the *SolutionPool* is emptied. W2RP-TD engine again runs to generate new solutions. This operation continues until the stop conditions are met: there is no more possible feasible solution to be generated, or the time limit has been reached.

3. The prototype tool in action - Screenshots

In this session we present one run-through of the W2RP-TD plugin on the ReleasePlanner™ project. The detailed project background and results will be analyzed and discussed in Chapter Six: and Chapter Seven:. The description and screenshot here serve as the proof of concepts for the tool and how it will conceptually perform in real-world projects. The project comprises a total pool of 29 features. The next release (at the point of this publication) is planned to be released after 80 person-days. There are 6 modules in total. We follow the plugin output (either via GUI or through console debuggers) in the steps of the work-flow proposed in Chapter Four:.

Step 1: Define TD criteria and metrics: This step is done implicitly via the parameters settings of TD. Currently, users can decide which range of release dates that can be varied, the level of TD acceptable in the system (e.g. 5% of effort can be invested in technical debt), and the expected defect detection rate in the system. These global settings can be further improved in future version of the plugin based on feedback of users.

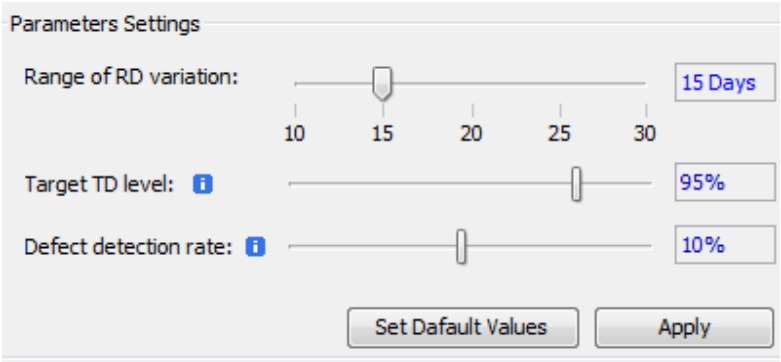

Figure 6: Global parameters settings

Step 2: Monitor and collect values: The business value of feature and the accumulative TD are collected and monitored via experts' opinion (via ReleasePlanner™ interface) and automatically based on issues tracking repository and automated testing metrics.

Features' business values are evaluated based on a set of criteria and experts' opinion.

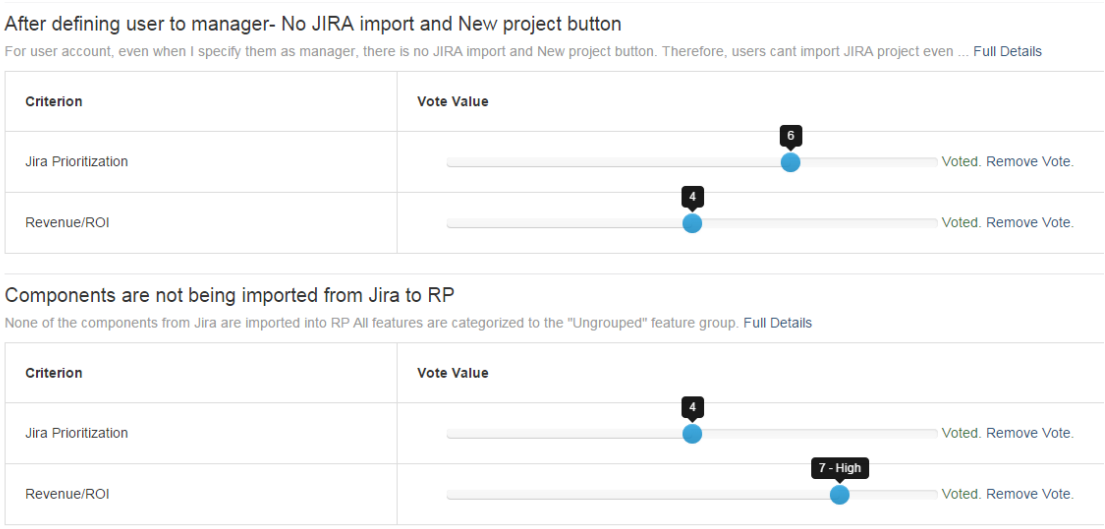

Figure 7: Features prioritization in ReleasePlanner™

Defects, enhancements, and issues are maintained via JIRA interface [36]. JIRA is a well-known issue-tracking tool that has been adopted in software engineering industry standard. Furthermore, ReleasePlanner™ provides standard interaction via JIRA API.

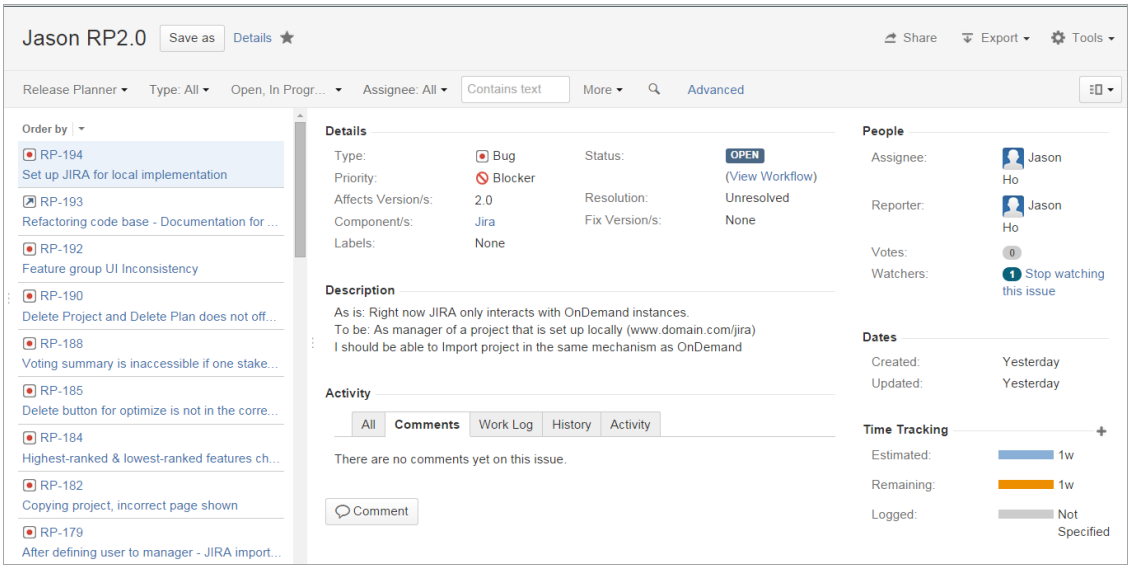

Figure 8: Issues tracking and management with JIRA™

Step 3: Estimate technical debt for the initial release date RD_0: the tool consolidates all the different metrics in TD and displays them in graphical representation, using charts

capability. The first Dashboard chart presents the trend of technical debt over time within a project. This is important for users in two aspects: (i) to inform users about the general trend of how TD accumulates within the project and (ii) which dimension of TD contributes the most towards the general trend. In the below example, we can see TD is on an upward trend, which is a potential risk. This risk mostly comes from increasing accumulated quality testing TD, which can be reduced via fixing of defects, and refactoring of code.

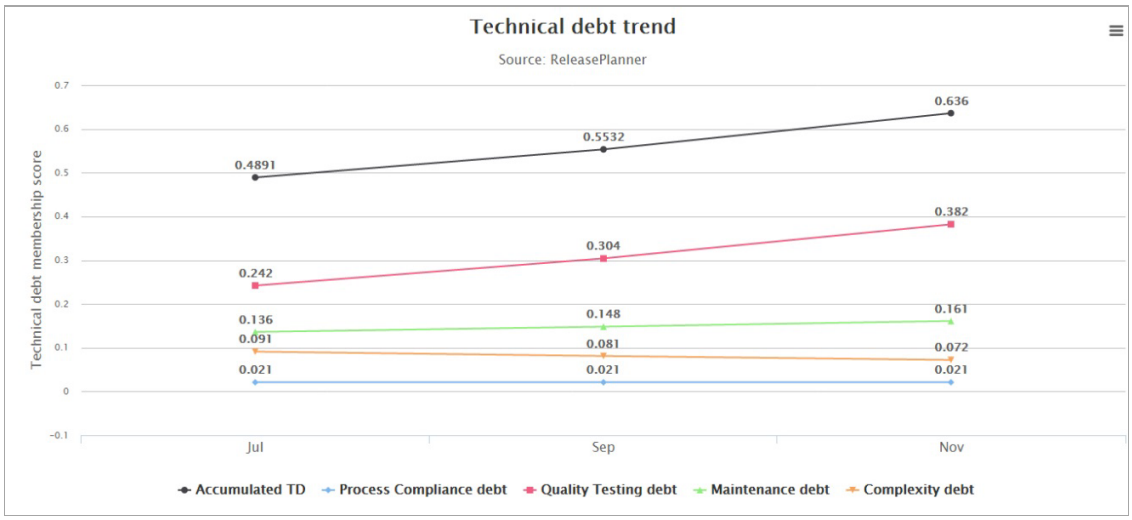

Figure 9: Technical debt trend over a period of release dates (RD)

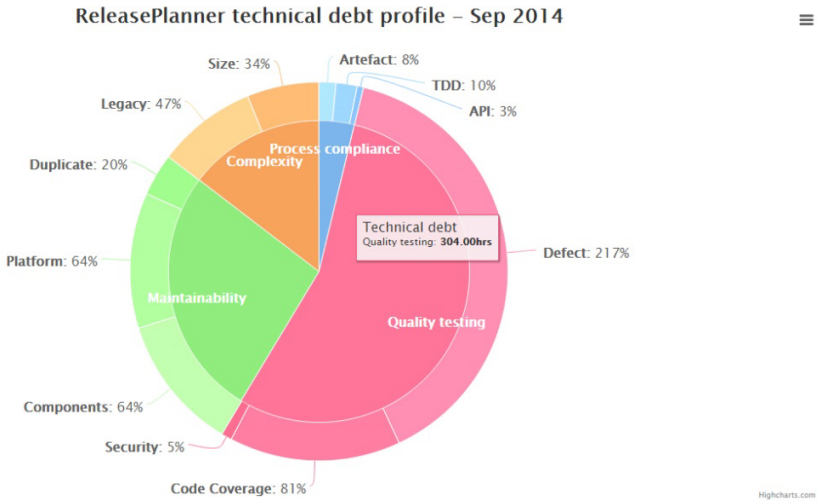

Figure 10: Technical debt breakdown

Users can also view the TD break down at a specific point in time. In this break down, users can view the distribution at the major level (the 4 main dimensions of TD) or drill down to the detailed breakdown of each dimension. Both screens are presented here under the same chart. Users can have the all TD in one chart option or the drill down option.

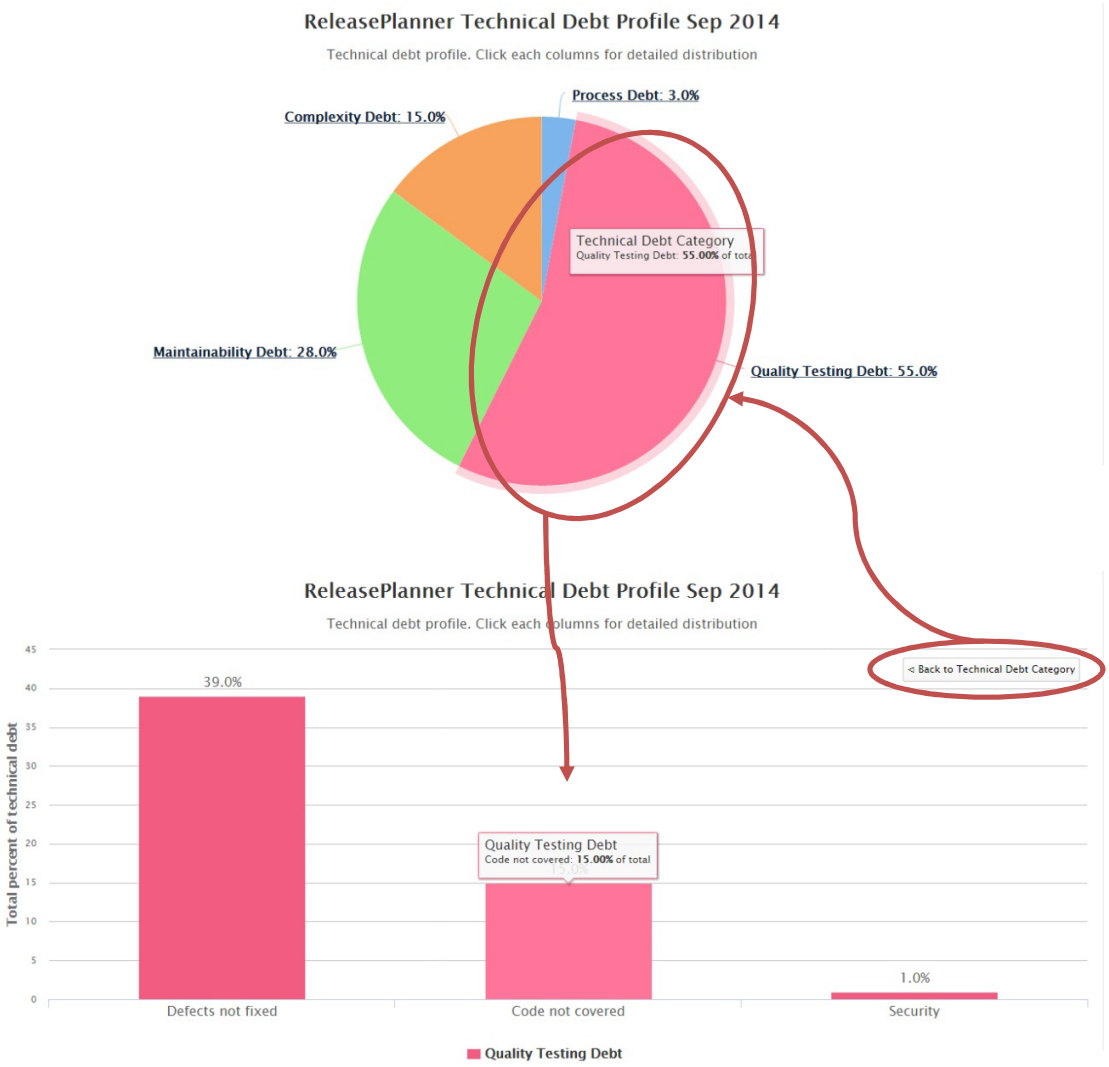

Figure 11: TD profile for RD_0

Step 4: Perform what-if analysis: user can interactively view the implications of varying the release duration, refactoring TD or modifying the set of features offered. Based on the global parameters previously set, users can change the range of their RD, TD or TRV on the provided sliders

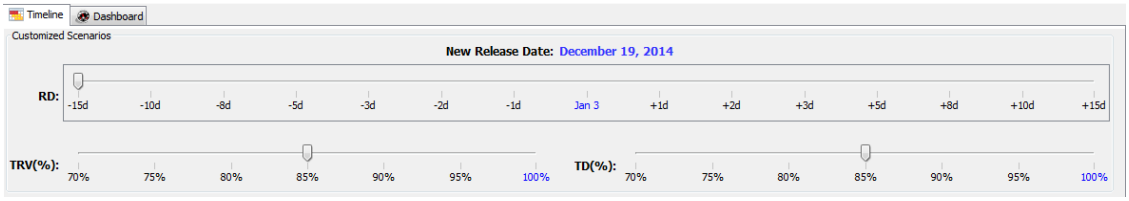

Figure 12: Interactive sliders to change the parameters to the desired level

Step 5: From all the scenarios explored, all trade-off solutions will be presented as candidate release alternatives, by *Analyze results*.

From the plugin engine, the local variation is executed for each ΔRD. The results in the log console are presented below:

Search Results | Output - WhenToRelease (run)

```
Unique R:-1.0 days      -1.0 days      136.0   97%
Unique R:-2.0 days      -2.0 days      136.0   95%
Unique R:-4.0 days      -4.0 days      136.0   93%
Unique R:-5.0 days      -5.0 days      136.0   100%
Unique R:-6.0 days      -6.0 days      136.0   84%

Remove R:-6.0 days      -6.0 days      136.0   84%
Unique R2:-6.0 days     -6.0 days      136.0   89%
Unique R:-7.0 days      -7.0 days      136.0   90%
Exclude1:-8.0 days      -6.0 days      136.0   87%
Unique R:-8.0 days      -8.0 days      133.0   95%

Remove R:-8.0 days      -8.0 days      133.0   95%
Unique R2:-9.0 days     -8.0 days      133.0   97%
Unique R:-9.0 days      -9.0 days      136.0   96%
Unique R:-9.0 days      -9.0 days      133.0   94%
Exclude0:-10.0 days     -8.0 days      133.0   97%
Exclude4:-10.0 days     -6.0 days      136.0   100%
Exclude4:-10.0 days     -10.0 days     133.0   97%
Exclude0:-11.0 days     -8.0 days      133.0   97%
Exclude4:-11.0 days     -10.0 days     136.0   100%
Unique R:-11.0 days     -11.0 days     133.0   91%
Unique R:-12.0 days     -12.0 days     131.0   100%
Exclude4:-12.0 days     -8.0 days      136.0   100%
Exclude0:-12.0 days     -12.0 days     131.0   100%
Exclude0:-13.0 days     -12.0 days     131.0   100%
Unique R:-13.0 days     -8.0 days      136.0   98%
Unique R:-13.0 days     -13.0 days     131.0   95%
Exclude0:-14.0 days     -12.0 days     131.0   100%
Exclude1:-14.0 days     -9.0 days      136.0   82%
Exclude4:-14.0 days     -14.0 days     133.0   97%
Unique R:-15.0 days     -15.0 days     131.0   93%
Unique R:-15.0 days     -11.0 days     136.0   84%
Exclude0:-15.0 days     -15.0 days     131.0   93%
TR SIZE:15
```

Figure 13: Textual results of solution pool

Graphically, these results are then displayed in a Trade-off solution tables, and presented on a bubble chart with the size corresponding to the level of remaining accumulated TD in the project.

Figure 14: W2RP-TD trade-off solutions are presented visually for experts' selection

Step 6: In *Select and re-iterate*, the product manager(s) can either go back to define another scenario or terminate the process and select the most attractive plan, representing the best balance between the criteria. When the users perform this action, the previous results are stored in the Dashboard. These results can be used as references in the future with the new What-if scenario. Furthermore, they can also be exported into spreadsheet format for future references and comparison.

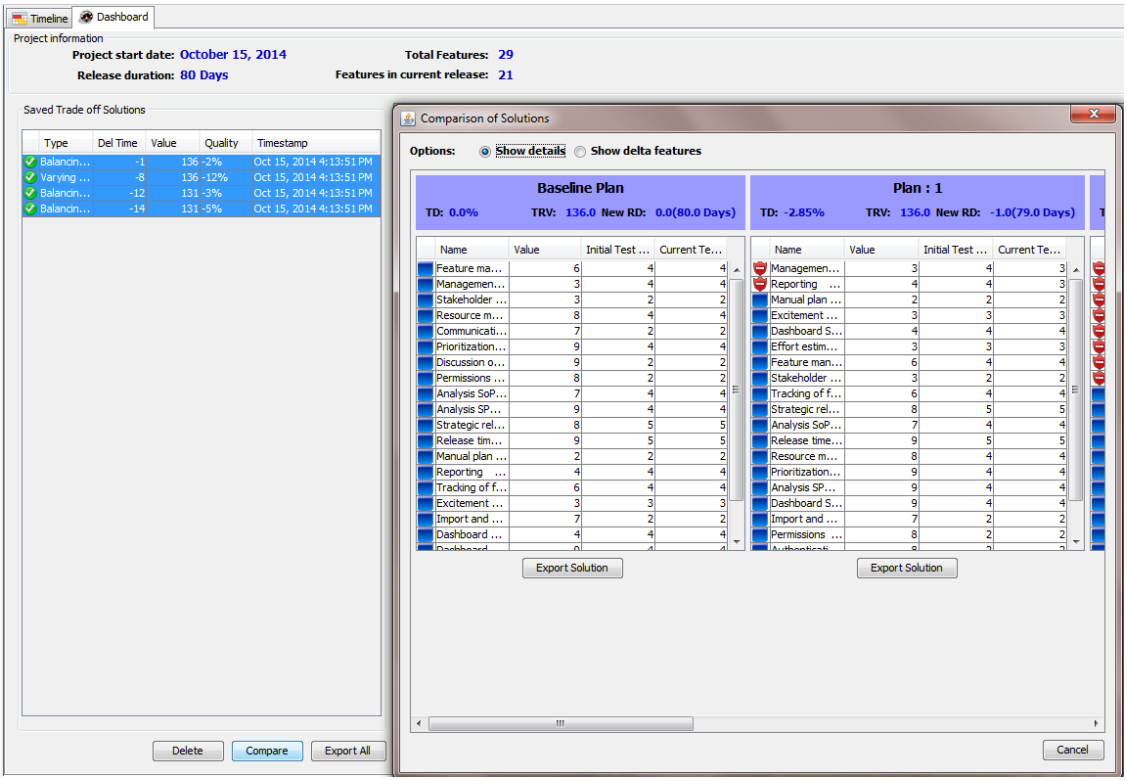

Figure 15: Saved solutions and comparison with different runs

Chapter Six: EMPIRICAL EVALUATION

1. Qualitative Interviews and Quantitative Questionnaires

To better understanding technical debt practices and validate the potential added values of the decision-making methodology and prototype tool, we conducted both qualitative and quantitative studies within software product organizations. Here in these studies we target both organizations with proprietary products, and Open Source Software (OSS) projects. Our objective is to gather as comprehensive understanding as possible in a variety of projects, domains, and maturity in technical debt management.

1.1. Study objectives and methods:

The following methods were chosen to explore the thought patterns and sharing patterns of our target users. There were three studies in total: (1) a retrospective data analysis, looking into technical debt practices in released product lines and OSS releases; (2) a quantitative survey was designed, consisting of 10 questions about effort estimation, quality management processes, and understanding of risk-benefits of technical debt; and (3) an in-depth interview with stakeholders in such organizations, from different perspectives: development team, testing team, and product management team.

Our goals in conducting these studies are (i) to gather the numerical data required to fine-tune the modeling of technical debt (TD) in our methodology, (ii) to validate the relationship between TD practices in relation to when-to-release planning (W2RP) decisions, and (iii) to perform trade-off the optimization tool W2RP-TD in real-time, complex projects. The general study objectives were then specified and validated for each case study, based on the context of the products and the organizations examined.

In the first case study, by studying publicly available data from OSS projects for two years, we learned the relationships (if any) between products expansion and releases with the accumulation of TD. To validate our result, we compared the findings in our

methodology with existing methods and tools support, such as manual track sheet, SQALE, and CAST. We then argued that industry-available tools do not appropriately measure and manage TD.

Following the first case study, we worked in collaboration with a well-established technology firm in Agile and automated-testing practices. Through studying the amount of TD accumulated in (both legacy and on-going) projects, we aimed to establish that if TD is managed appropriately using our methodology and tool support, when-to-release decisions can be optimized to maximize release value and minimize TD.

Lastly, after establishing the potential usefulness of the method, we applied the method and the tool on a proprietary project with business pressure to deliver in two-years, yet TD has been accumulated to a state of hindering features-development. We validated the usefulness of the methods with the project manager and stakeholders of the organization.

The designs, data collection, and findings of these case studies are explained in the sub-sections that followed.

1.2. Case study methodology

In order to measure the effectiveness and efficiency of the methods in real world, complex projects, we conducted case studies in (software) products with different nature in scope, functionality, and development teams. Case study is a suitable research methodology for software engineering research since it studies contemporary phenomena in its natural context [84]. In each case study, we deployed three different research methodologies to investigate and validate the approach we proposed as part of this methodology. These methods include: retrospective data analysis, semi-structured interviews, and surveys. Each will be discussed in details in the next sections.

1.3. Retrospective data analysis

Retrospective data collection and analysis is useful at the beginning of the project to understand the current state of practice in technical debt management, and learn potential recommendations for improvements. The retrospective aspect is important as technical debt is a relatively recent concept, and therefore not much data has been previously available [42]. Furthermore, if we can reflect on past phenomena with the proposed methodology, we can compare the outcome prescribed by the method with the actual outcome of what happened in real-life projects, to validate (or invalidate) the assumptions as well as findings in this process.

The summary of the retrospective data study can be found in [35]. We selected five (5) OSS projects and compare them over the duration of 24 months in the domain of desktop, mobile, and web applications. These systems were representative of the type of software projects that could benefit from our planning methodology. Furthermore, they had enough wealth of publicly available data for retrospective comparison in our case study. Below, in section B.1, is a summary of the projects that have been selected, and the highlighted releases that we examined further for W2RP-TD decisions.

1.4. Design of interviews

Interview is a commonly used technique for collecting qualitative data [87]. We conducted semi-structured interviews to achieve the above mentioned objectives. Our target audiences were product managers, project managers, testing team and developers. We aimed to analyze the opinions and impressions from the software project team from different perspective to understand TD top-down and bottom-up. Through interviews, we wanted to identify the awareness, understanding, and perception of real-world project teams with regards to technical debt management. The general questions were prepared prior to the interviews and sent to the interviewees. However, these served as a guideline

for the conversations, where further in-depth follow-up questions were asked for deeper understanding. The structured part of the interview aimed at the definition and monitoring of technical debt, while the follow-up questions are behavioral oriented in how these team members made decisions.

The interviews aimed at understanding in-depth the perception of technical debt. All questions were designed to understand the work process, starting from the day-to-day operations, taking into account that they are specific perspectives and may not be a complete representation as a whole. Whenever possible, we posed quantitative questions such as time spent on certain task, or distribution and prioritization of tasks, or the cost or value they would bring. This revealed interesting aspects of how product and project teams made when-to-release and technical debt trade-off decisions in real-time.

Lastly, there were three sets of questionnaire, each aimed at different target audiences. The purpose of this differentiation was two-fold: (i) to identify the similarity and dis-similarity about how different teams perceive and measure their TD and (ii) to understand holistically (before determining decision-support actions) at the organizational level the when-to-release decisions in consideration of TD.

The interview questionnaires are enclosed in Appendix B.

1.5. Design of surveys

Surveys are often used when various method(s)/tool(s) have been used in an organization (or group of organizations) [54]. In our studies, the users of the method(s)/tool(s) were asked to provide information about their understanding of technical debt, the weights of each criteria being listed and their approach in measuring the risk and benefit of technical debt. With this information, we could map the quantitative data into our modeling, and potentially identify any missing gaps from our understanding. The survey further assisted us in getting to broader audiences from different teams and different projects. This

technique is used to evaluate the differences between method(s)/process(es) with respect to the management of technical debt.

The surveys were designed to understand (i) quantitatively the effort required to fix technical debt, and effort required to accomplish other activities besides from technical debt, and (ii) qualitatively the operations of software development team, and the process of managing technical debt and when-to-release decisions (or the reasons for the lack of such decision-oriented processes).

With the metrics proposed in Chapter Three:, we design the studies in order to achieve the following goals:

Table 4: Goals of qualitative and quantitative questionnaires

Goal/Questions/ Metrics	**Positive impacts**	**Negative impacts**
Addressing technical debt	- Hours to understand and identify root cause of an issues (that are otherwise not well-documented) - Change Management Control process oriented - Hours saved to re-design automated testing - Hours to execute the test suite again after change	- Hours required to fix a bug (classified by severity) - Hours spend on refactoring/improving code that are not defects - Hours to design/execute manual testing

Goal/Questions/ Metrics	Positive impacts	Negative impacts
Not addressing technical debt	- Hours saved on not needed documentations (trivial features, minor bugs) - Hours saved on deferred bugs/change requests - Revenue of features offered in place of non-feature effort (classified by complexity/class) - Cost of opportunity if a feature/change request is not offered	- Hours/cost if a defect is found after released, or discovered by users/clients - Hours of freeze, damage cost in case of a catastrophic event - Impact on Test run-time (slow, extended, hang, etc.) - Impact on code complexity (SLOC, legacy, etc.) - Hours to document missing documentation after discovery that it is required

The detailed questionnaires are included in Appendix B. Below is the summary of the data collection process, followed by findings and analysis from the gathered data based on these conducted studies. The surveys used in this thesis provide a more detailed look at TD, compared to a similar work that has been conducted in Finnish-European context [52]. The final version of the surveys is included in Appendix C.

2. Data Analysis and Findings

The case study applied the methodology, both in current releases and retrospectively in past releases (when available), to compare the value of the method (and the prototype tool) in the following criteria: (i) the degree of increase in visibility and in management of technical debt (TD) by comparing the availability of information, and the efficiency of the team in maintenance and implementation of new functionality after release and (ii) the

superiority (or lack thereof) in the portfolio of solutions available for decision-makers, evaluated by human experts, when available.

We present below three different case studies from different organizations with very different organizational, products, and projects background. The three cases investigated are selected based on their diversity in scale, team size, phases of development, and availability of stakeholders and human experts. We will then determine in which scenario the method is most useful, and the rationality behind their success (or lack thereof). The purpose of this selection is to ensure we eliminate any biases that may be introduced due to organizational specific conditions and to (potentially) replicate and externalize the results of the methodology.

2.1. Case Study 1: Open-source Software (OSS) projects

In a research paper published at ESEM 2014 [35], M. Felderer et al. reported about a case study for applying systematic planning methodology to perform quality-driven releases. Based on the analysis of effort estimates, defects and features, retrospective analysis showed how early investments into quality can improve later results. Improvements were demonstrated in terms of reduced release time, more features implemented in the same time frame, and better overall release quality (less defects slipping through). This research was the starting point for us to investigate further into other OSS projects with data available. We aimed to first potentially replicate the result of this initial survey into other projects, and potentially investigate if technical debt management can make a positive impact to when-to-release decisions.

2.1.1. Study objectives

For this case study, given the above objectives, we specified the goals as followed:

Context: Given products that are open source, with many contributors, and publicly available source code and releases information. In the period of two years of evaluation,

Objective 1 (O1): As projects expanded in size and complexity (regardless of domain or platform they are in), we formulated and monitored the trend of TD, in all components. We then evaluated if TD was currently tracked or managed appropriately using existing tool support in these projects.

Objective 2 (O2): From earlier results, we learned that when-to-release decisions are made largely without consideration of TD. We aimed to validate that, with the growth of TD, maintainability deteriorate (in term of issues faced and productivity loss), causing longer release cycles.

For OSS projects, human experts and developers were difficult to reach and when-to-release decisions were not obvious made public. Therefore, O2 was meant to partially establish the relationship between TD and W2RP, indicating that a methodology, such as that proposed in this thesis, might reduce or eliminate the loss in productivity and delay in release plans.

2.1.2. Organization background

The projects were selected based on their development cycles and release duration. We needed a consistent development period to compare them. They also needed to have the same range of major releases for us to have enough data for analysis. Lastly, we selected OSS in different domains and marketplaces, to avoid potential dependency on a specific industry. The summary of the selected five (5) OSS projects and their domains are as followed:

Table 5: OSS Projects to be considered for study in TD measurement and trade-off

Name	Releases Information				
Log4J	Major releases: 2. Minor releases: 4				
Source:	http://logging.apache.org/log4j/2.x/download.html				
Releases (Date)	Jun-2012	Jan-2013	Jun-2013	Jan-2014	Jun-2014

Name	Releases Information				
Version	2.0.alpha1	2.0.beta4	2.0.beta7	2.0.rc1	2.0.0
Publify	Major releases: 8. Minor releases: 13				
Source:	https://github.com/publify/publify/releases				
Releases (Date)	Jun-2012	Jan-2013	Jun-2013	Jan-2014	Jun-2014
Version	6.1.0	6.1.1	7.1.0	8.0.0	8.0.2
Github Android	Major releases: 1. Minor releases: 13				
Source:	https://github.com/github/android/releases				
Releases (Date)	Jun-2012	Jan-2013	Jun-2013	Jan-2014	Jun-2014
Version	1.0.0	1.6.1	1.7.1	1.8.0	1.9.0
Highchart	Major releases: 4. Minor releases: 24				
Source:	https://github.com/highslide-software/highcharts.com/releases				
Releases (Date)	Jun-2012	Jan-2013	Jun-2013	Jan-2014	Jun-2014
Version	2.2.5	3.0.beta	3.0.3	3.0.8	4.0.3
jRuby	Major releases: 2. Minor releases: 22				
Source:	https://github.com/jruby/jruby/releases				
Releases (Date)	Jun-2012	Jan-2013	Jun-2013	Jan-2014	Jun-2014
Version		1.7.2	1.7.4	1.7.10	1.7.13

Although we could publicly access the source code, the challenge with these selected OSS projects was that we did not have access to the human experts and the decision-making process internally. Therefore, to validate our study result, we compared the releases and accumulated TD estimate with two different sources:

- We compared our TD tool with the existing plug-in tools following similar estimation method such as the SQALE [60] or CAST [67] methods, as highlighted in background and related work chapters
- We compared our TD estimates and trade-off recommendations with the actual release process (through retrospective data analysis). We argued that, given the previous releases and stages of the OSS projects, we could compare the recommendations made by our tool with the actual outcome of the status of the releases and the existing TD in those projects.

With this goal in mind, we aligned and analyzed the product based on 4 different cycles, each lasting 6 months. 6 months were chosen as the release cycle for several reasons:

- Different OSS projects had different release cycles. However, in approximately every six month, there was a new release. In business sense, it could be explained as the mid-year and year-end strategy of releasing software.
- This was a long enough duration for (significant) changes in implementation to be realized in the OSS projects and TD tracking.

2.1.3. Product background

In this section, we detailed the products being chosen for analysis and why we picked them for this study. When searching for the products, we looked for the wealth of available information, as well as the duration and releases of development, to ensure retrospective data analysis can be helpful in predicting the applicability of the methodology. Lastly, diversity in domain and functionality was important. We aimed to discover any dependency of the TD management practices with the organization operations.

The five products examined are:

- Log4j (http://logging.apache.org/log4j/2.x/): is a popular logging service provided by Apache. The product has gone through two major releases; with version 2 addressed major functionality change to catch up with competitor (Logback) and multi-threaded asynchronous logging mechanism. This product was chosen due to the historical state of the product. Furthermore, the project teams used SQALE Sonar to track their TD previously. Having this data as a form of comparison was helpful to our research.
- jRuby (http://jruby.org/): is an implementation of the Ruby programming language on Java Virtual Machines (JVM). The product was of interest as it has two dependencies on programming languages, on Java and on Ruby. Recent releases included Ruby version upgrade (from v.1.9.x to 2.0) support. This product could illustrate how external elements such as a release of a new language version could impact the internal TD and implementation of a team.
- Github (Android platform) (https://github.com/github/android) is the Android platform development of the popular web-based service Github. Github is a Git repository web-based hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. GitHub provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project. [98] This project was chosen as it is a Mobile platform project, incorporating the fast development and deployment life cycles of Mobile platform applications, in this case based on the Android platform.

- Highcharts (www.highcharts.com): is an interactive Javascript chart utility that is used by many web applications. The product is publicly available with options to upgrade to paid membership. The applications are in the visualization and charts domains. Part of the plugin interface in this thesis was built using this technology, making it a good choice for analysis in TD. Furthermore, as there is a diverse set of chart options, TD estimates and release planning became more complex, interactive decisions with development team.
- Publify (http://blog.publify.co/) is an open-source blogging service, utilizing Ruby on Rails engine with extended publishing capacities. Publify and supported plugins are free software released under the MIT license. The project was chosen as it is a user-centric application with extended documentation and releases notes for their release process. The user-centric nature of the blogging service also provided interesting insights on how the team address and account for future development with the underlined notion of TD (users' feature requests, platform upgrade, etc.).

2.1.4. TD Analysis

Based on our workflow proposed in Chapter Four:, Section 1, we needed to first determine the GQM of TD. In this case study however, as the projects are OSS, and the human experts are not available, we decided on a set of common metrics that are used to measure TD, both from literature reviews, and from other methods of measuring TD, such as SQALE or CAST.

The SQALE method [38] [60] measures TD as a composite of different categories, coupled with their set of metrics. The main components of TD are measured in 7 –ity: reusability, portability, maintainability, security, efficiency, changeability, reliability and testability. These factors are in turn measured by metrics such as duplication, violation, comments, coverage, complexity and design. A summary of TD size and trend in the OSS

projects are included in below. Generally, from observation, as the code size increased, there is a general upward trend of TD, as measured by Sonar.

However, as analyzed previously in [58], methodology that attached the dollar figures to compliance such as SQALE potentially lead to systems having more estimated accumulated debt than operation cost or operational profit. In the case of jRuby, the effort to reduce TD was estimated to be more than 2000 person-days (more than 6 person-years). For log4j and Publify, TD was more than 180 person-days (more than 6 person-months). While these indicators indicated these OSS projects were ridden by growing accumulated TD, it was less meaningful in term of management and reduction of TD. Furthermore, the large values in three of the above mentioned projects might discourage managers from even attempting to manage TD to begin with.

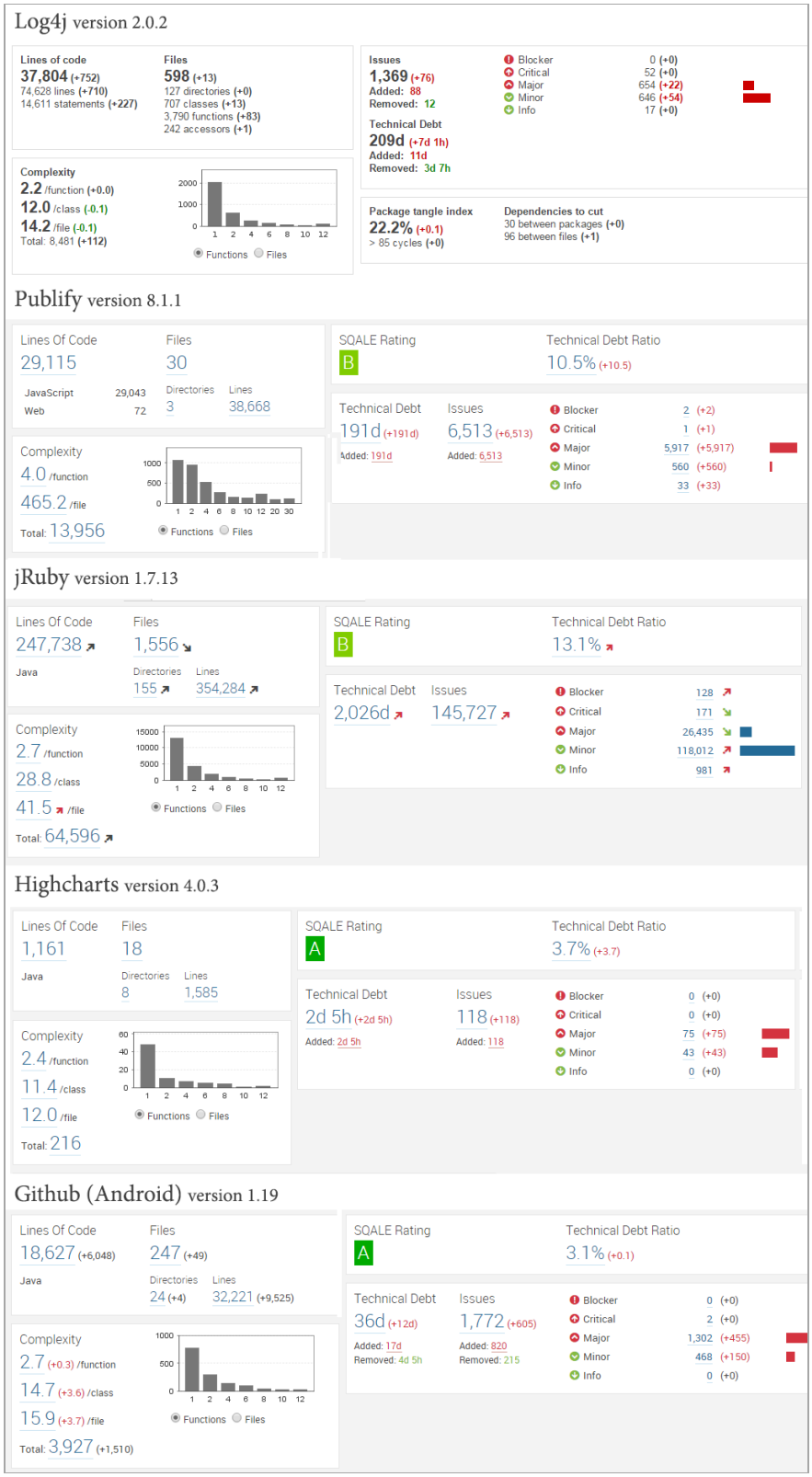

Figure 16: TD trend for all 5 OSS projects, as measured by Sonar plugin

In the CAST method [25], the authors propose a formula for TD principle estimation using cost based analysis of violations (issues, defects, etc.) using their triage and severity level: high, medium, or low. The method assigns a cost and time required to fix these violations (according to their severity). In this case study, we used the default values as proposed by the method [25]. *Table 6* shows the issues count, and their severity in all of 5 OSS projects. In addition to the measures proposed by CAST, we also counted the aging issues, issues that are not fixed for more than 90 days and 180 days, to show the degree of accumulated debt in the systems.

Table 6: Issues count and their severity in all 5 OSS projects over 2 years

Projects	Critical	High	Medium	Low
jRuby	0	128	171	26435
Publify	0	2	1	5917
Github-Android	0	2	1302	468
log4j	10	124	46	7
Highcharts	0	75	43	0

Based on this information, the TD trend, as calculated by CAST, is presented in the table below:

Table 7: TD estimates for 5 OSS projects, as of Sep 2014, by CAST method[4]

Projects	Accumulated TD as of Sep 2014
jRuby	$2,046,675
Publify	$444,525
Github-Android	$231,000

[4] Using default values as proposed in [67]

Projects	Accumulated TD as of Sep 2014
log4j	$44,625
Highcharts	$14,475

Once again, the cost of TD, as measured by CAST, was significantly large. An OSS project such as jRuby, Publify or Github would be unlikely to afford hundreds of thousands of dollars for TD activities.

Discussion of the SQALE and CAST method is outside the scope of this case study. The methods are presented as part of related works in Chapter Two:, Section 3.2. As a mean of comparison, we combining the list of these metrics with our proposed GQM and focused on the following overlaps:

Table 8: GQM for the evaluation of OSS projects

Dimension	Criteria	Metrics	Weight
Process Compliance	Availability of Interface (API)	The percentage of documented interfaces.	w_{d1}
Quality testing	Reliability	Number of defects remaining at point of evaluation	$w_{d2\text{-}1}$
	Code Coverage	The percentage of code not tested, or not covered by automated tests	$w_{d2\text{-}2}$
Maintainability	Components coupling	The interdependency between components	$w_{d3\text{-}1}$
	Reusability	The modular degree of code, the percentage of duplicate code	$w_{d3\text{-}2}$
Complexity	Size	Current system size	$w_{d4\text{-}1}$

Dimension	Criteria	Metrics	Weight
	Complexity	Current system code complexity	$w_{d4\text{-}2}$

In step 2, we evaluate the membership score of each of these criteria using the available data collected from Github, JIRA and the source code of all the OSS projects. Below was the result for log4j project (all other projects data were available as well)

Table 9: TDAF calculation for log4j project

Dimension	Criteria	Weight	Fitness Score
Process Compliance	Availability of Interface (API)	0.25	0.05
Quality testing	Reliability	0.125	0.22
	Code Coverage	0.125	0.13
Maintainability	Components coupling	0.125	0.22
	Reusability	0.125	0.01
Complexity	Size	0.125	0.3
	Complexity	0.125	0.1
TDAF			0.135

Based on the membership score, the TD estimate is calculated to be 1.63 person-months (36 person days), or 1.16% of the total code base. The distribution of TD is visualized in the Figure below:

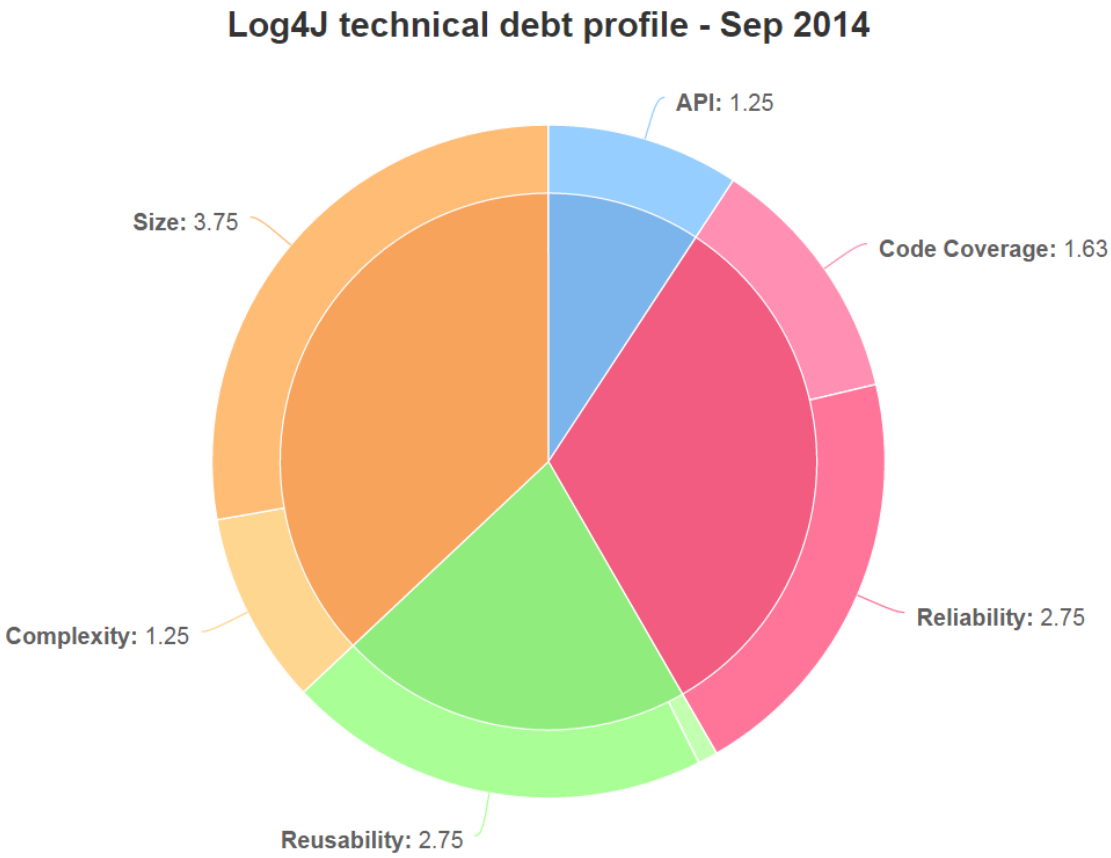

Figure 17: TD profile for log4j OSS Project

Comparing with Sonar and CAST method, technical debt (TD), regardless of the different criteria that define it, consistently grew over time as the projects grow in size and complexity. Some of the measures we used were similar to that of industry standards. However, by using the percentage of estimated effort to address TD, arguably, our method provided a more manageable approach to address TD as part of the total release process. The figures we presented were also more conservative and tangible compared to that of Sonar and CAST.

a. Recommendations and Outcomes

In this case study, decision support is not available as the project is open source and release information is retrospective. However, by analyzing retrospectively past releases, we establish (peripherally) the relationship between release quality, the point of release,

and the (accumulated) TD in these projects. As established in previous section, TD indeed grew in these OSS projects (across different evaluation methods).

We examined the 'aging' of issues in these systems, both 'Open' and 'Closed' issues. Accumulated issues that are not fixed for more than 90 and more than 180 days are tracked. These issues signaled that systems are getting harder to maintain, and defects are taking longer to be fixed. In all systems, there are close to 5% or more issues that remained opened, or only get addressed, after 180 days (6 months – more than a major release). Postponing issues and fixes is also identified as one of the most major cause of TD in systems.

Table 10: Issues aging in 5 OSS projects considered

Github-Android							
Aging	< 7 days	< 30 days	< 90 days	< 180 days	> 180 days	Total	
	90	30	22	10	5	**157**	Closed
	57.32%	19.11%	14.01%	6.37%	3.18%		
	11	26	41	26	81	**185**	Open
	5.95%	14.05%	22.16%	14.05%	43.78%		
jRuby							
Aging	< 7 days	< 30 days	< 90 days	< 180 days	> 180 days	Total	
	822	213	128	70	58	**1291**	Closed
	63.67%	16%	10%	5%	4%		
	0	252	153	136	45	**586**	Open
	0.00%	43%	26%	23%	8%		
Highcharts							
Aging	< 7 days	< 30 days	< 90 days	< 180 days	> 180 days	Total	
	424	66	52	30	10	**582**	Closed
	72.85%	11%	9%	5%	2%		
	145	6	14	39	14	**218**	Open
	66.51%	3%	6%	18%	6%		
Publify							
Aging	< 7 days	< 30 days	< 90 days	< 180 days	> 180 days	Total	

	244	34	36	46	16	**376**	Closed
	64.89%	9%	10%	12%	4%		
	1	16	23	12	29	**81**	Open
	1.23%	20%	28%	15%	36%		
Log4J							
Aging	**< 7 days**	**< 30 days**	**< 90 days**	**< 180 days**	**> 180 days**	**Total**	
	43	76	102	46	379	**646**	Closed
	6.66%	11.76%	15.79%	7.12%	58.67%		
	42	35	38	27	42	**184**	Open
	22.83%	19.02%	20.65%	14.67%	22.83%		

In term of when-to-release, or quality of release decisions, we analyzed past release-notes, publicly available forums, and code reviews for each project. In the case of log4j, it took the team 2 full years to finish alpha and beta for their major release 2.0. For Publify, the main developers announced formally release 8.0 as the "biggest release in 9 years". Highcharts announced a release announcement with complete graphics for release 3.0 back in 03-2013, and a year later version 4, in 04-2014. For the last two cases, the releases are more often with mostly bug fixes and enhancements. However, it is noteworthy that Github Android is a mobile apps with a different update-release methods, and jRuby is a platform independent JVM.

One last key element to look at was the contributors into these systems. We predicted that, as open source projects, they may have a large stream of people working and contributing to the source code, and the releases. However, this assumption was not true. Figure 18 below outlined an example pf the contribution profile of users in master code

branch. Only 1-2 major contributors actually made major impact to the releasing of the code, in the case of jRuby. This was consistent with all other projects.

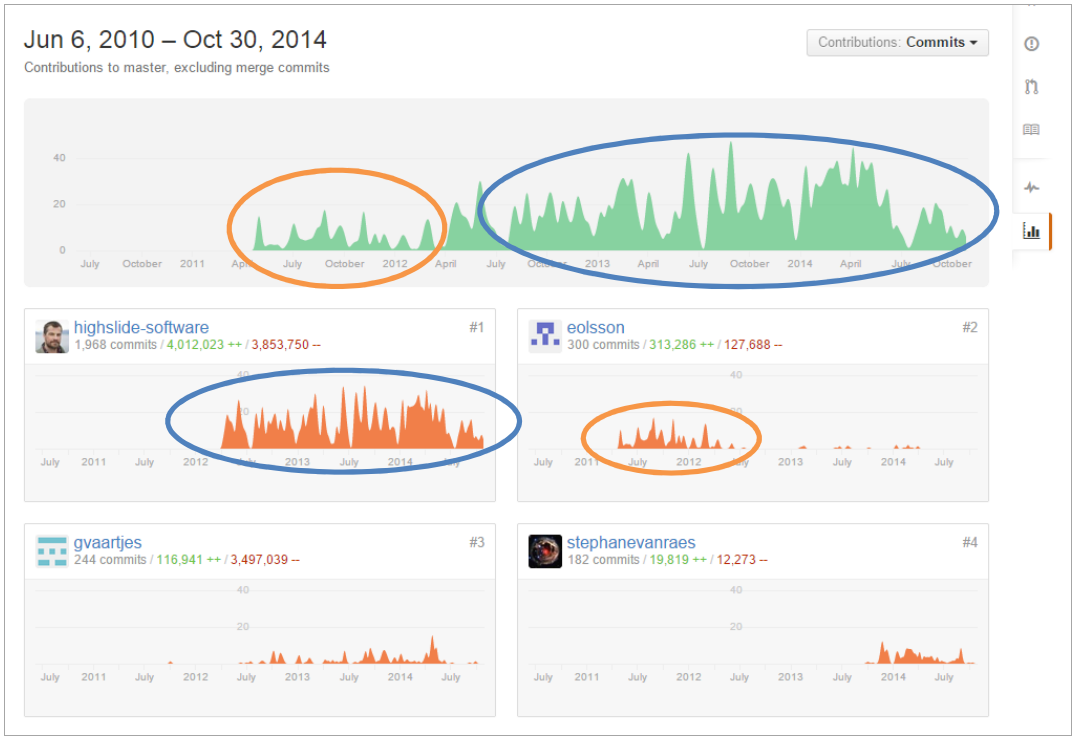

Figure 18: Code contributors into Highcharts project

2.2. Case Study 2: SMART Inc.

This study was completed as part of an ENGAGE collaboration between SMART Technologies (http://www.smarttech.com/) and University of Calgary. The project initially aimed to optimize automated testing, which has been successfully adopted at SMART. During the course of the study, through interaction with stakeholders and team members, via interviews and surveys, the team recognized the importance of technical debt (TD) in the projects. In this thesis, we aimed to learn more about TD in a mid-size organization, with organizational level adoption of Agile methods and well-monitored testing metrics. With that knowledge, we attempted to apply our methodology to define, monitor, reduce and potentially incorporate TD into release decision support.

2.2.1. Study objectives

Context: Given a mid-size to large scale organization with projects lasting more than 12 months, with teams collaborate well in Agile-driven environment and experienced automated testing practices,

Objective 1 (O1): We studied the trend and size of TD in both legacy and on-going projects. We planned to evaluate the existing methods and tool-support for quantifying TD (if any). We then compared the results with our modeling for analysis.

Objective 2 (O2): We evaluated the potential that if TD was managed appropriately (implicitly by following our proposed methodology) on an organizational level, the impact to release decisions (both what to release and when to release) could be optimized quantitatively and beneficial to productivity and release duration.

2.2.2. Organization background

SMART Inc. (http://www.smarttech.com/) is a leading technology and innovation company with the desire to improve the way the world works and learns [103]. Through innovative products and development, SMART assists its customers – the teachers, students and business people – to discover new ideas, new ways of learning and new ways of collaborating. SMART's major products include easy-to-use interactive displays and integrated solutions that are crucial to classrooms and meeting rooms around the world.

As part of the initiative to implement better products in shorter time to market, SMART has adopted organization-wise automated testing to maximize test coverage and products quality. Automated test execution is one of the most popular and available strategies to minimize the cost for software testing, and is one of the central concepts in modern software development as methods such as test-driven development gain popularity [97]. However, with the business pressure of ever shorter release duration,

there is a rising concern that larger code coverage alone will not result in less defects and less technical debt during the development process.

This particular study examined real world practice of automation in testing when coupled with manual defect testing, in consideration of technical debt. We examine retrospective data, coupled with stakeholders' interviews and surveys, to determine if quantitative optimizations could be made to decrease automated testing run time, increase code quality, and decrease technical debt.

2.2.3. Product background

The main products of SMART Inc. are SMART Boards ® hardware and collaborative software. There are multiple software products that support the hardware sold by SMART. The display boards are both interactive and collaborative by nature. In the study conducted with SMART [46], we examined specifically their core products, which are SMART amp and Whiteboard platform (known as 'core' internally). The project has been developed for the past 2 year.

From the product team, there are two program managers, with 10-12 developers in testing team and development team. The project followed a well-designed, efficient three-week sprint cycle of development. Releases were made frequently, either quarterly or bi-annually. The program managers traveled to collect requirements from key partners, educational conferences and program, and actual users experience review. The requirements were prioritized between program managers either based on criticality of major customers, or by projected revenue such features will bring.

From the development point of view, at the beginning of each iterative development cycle, the testing team met with the development team to review the use case scenarios. The team collaborated and decided collectively which user stories (sub-features) and by which team member they would be implemented.

2.2.4. State of development

The interviews and the surveys have been approved by an ethic review process by the research ethic board (Reference ID: REB14-1245). This interview was done as part of an NSERC ENGAGE project to optimize technical debt and development process at Smart Inc. Three interviews were conducted with three interviewees, each representing their team: program management, testing and development. Each interview lasted about 45 minutes with the same set of questions, exploring current processes and TD management techniques. The interview questions are included in Appendix B of this thesis.

All our participants from product management, development, and testing teams work on the same core product that was highlighted above, i.e. Smart amp® and whiteboard platform (aka. core). The answers were recorded, with participants' consents, as raw data analysis. Below were the major findings from the combined data.

From the process point of view, project and features priorities were decided by collaborative sprint planning. There were two program (product) managers, a business product manager and a functional product manager. They collected data and prioritized the features based on available projected revenue. There was a large amount of data available from multiple sources. These sources were teachers-students from education programs, developers' interview, conferences, summits, and premium customers' requirements. However, all the features and users stories were tracked by spreadsheets, without systematic tracking or prioritizing.

In term of development, SMART adopted Agile methodology-Scrum based iterative development, with each sprint cycle of 3 weeks. The process was a very well-managed Agile process, as measured by Agile maturity model [74]. Typical sprints were structured as:

- Week 1: sprint planning, story structure, effort estimate, features assignment.

- Week 2: development, testing, fixing (if there are defects).
- Week 3: ready for release pipeline.

The development pipeline was structured in this other:

1. CI Env (sandbox): a 'playground' for developers.
2. Dev Env: developers and testers code environment.
3. UAT Env: most important environment for all testing activities, where automated tests are run, coupled with manual explorative testing.
4. Pre-production Env: as close to Production as possible. This is where the last safety-net tests are run to ensure no breakage in production.
5. Production Env: the product environment to be used by clients and customers.

There was a very close collaboration between testers and developers. In the event of a defect, there was direct, local communication to identify root cause together to fix the issue. The turnaround time was relatively quick (30 minutes – 1 hour) to identify root cause. There was also a good code review process to avoid code duplication.

Although not happening very often, there was a context-switching cost of tasks switching, estimated at couple hours. This occurred one or two times every sprint in the event of an unforeseen defect. However, escalation or conflict of interest was not mentioned by participants, either there was little divergence or participants were not comfortable talking about such disagreement.

2.2.5. Automated testing

In an industrial survey, 88% of software development companies did not use fully automated test systems, opting for less reliable manual testing instead [22]. However, full automated testing was neither achievable nor recommended. In some cases, automated testing could potentially increase TD [97].

At SMART, test automation was generally received positively, mostly thanks to the fast detection of change, and the thoroughness check of code base. However, the test suite took at least 2 hours to run (and had to be capped at 4 hours). Test suites took "too much time to code", and "generally feels like high-level unit tests" [46]. Automated testing was integrated heavily in development and testing. The testing development lead indicated that "our testers typically spend 75% on writing automated test scripts".

All test logs were written in human readable language, to facilitate communication, ease of documentation, training for new testers, and collaboration with product managers. The automated testing process only started in about 12-14 months, as indicated by program manager. There was an organizational mandate of having a minimum 80% of test coverage. However, this figure was perceived as superficial by team members interviewed. Not everyone pushed to achieve better coverage due to the lack of metrics and guidelines, inability to detect breakage effectively, and lack of direct impact to internal code quality.

Furthermore, as the interactive board required simulated user-input and activities, the automated test suites often only accomplished many basic, "almost unit-test-like" functionality. There was a potential issue with test suites size: Automated tests were added constantly whenever there was change; yet, no automated tests were removed, no matter how long ago it was added, or it had not failed for a long time. Potentially, this would translate to a potential problem of too large test suites. Currently, the solution was to "throw more hardware at it", dividing the suites into smaller chunks that run in parallel in testing environment.

Lastly, there was currently no estimate process. There was no track record of similar stories for future reference. Developers and testers tracked their own timeline. Generally, there was no estimate process for development, and no feedback loop for previously identified issues.

2.2.6. Technical debt analysis

At SMART, the teams were generally aware of technical debt (TD). However, there was no (set of) metrics to track effectively. This lead to the lack of a particular priority: teams dedicated time to address TD, without knowing how much of the total accumulated debt there was. In a rough estimate by team members during interviews, TD effort was around 10%-25% of the total effort spent in each sprint. To program managers, for every milestone in the product pipeline, there was a sprint for reducing TD or internal improvements. Technical debt came from a discrepancy in development and testing:

- In testing team, automated testing followed 80-20, and covered all required scenarios
- In development team, there seemed to be a 'too trivial to test' for some unit testing on functionality

Each team had different perspective on TD and how they addressed debt:

- From the Project Manager point of view: At least there was one "quality (QA) sprint" every quarter. The business program manager described this sprint as "spit and polish" process. For them, it was their effort to meet big milestones that caused the most TD. However, the PMs were confident they addressed them *all* in one QA sprint.
- In testing team, there was a test-operation-developer (TestOps) who dedicated his time and effort to fine-tune and improve automated test suites. In testing, TD accumulated from growing automated suite, and slow runtime. Estimated, less than 10% effort was spent on TD in testing activities.
- In development, there was typically no feedback loops. So even if catastrophic or rare defect happens, no learning was done. It was estimated 15% of time is spent on 'code refactoring', which was, to developers, the main component of TD.

- In term of prioritization, 80-20 rule applied. The teams fixed and cleaned up important issues or features first, and ensured at least 80% efficiency.

With this initial understanding, we designed the surveys in section A.3. It was then sent to participants within the organization to different teams: product management, developers, and testing team. There were 10 questions in total (Appendix C). There were two rounds of survey, with round two including slight modifications to improve the understandability of the questions based on the feedback from the first round. The first survey was distributed in a period of 3 working days, and there were 34 valid responses in total, with 18 participants from testing team and 16 participants from development team. The second (revised) survey received 48 responses, 10 from testing team and 38 from development team. 21 participants in the second round of survey were first time participants.

On average, the participants surveyed were aware and acknowledge that there is TD in the projects that they work or have worked in. Typically, test case designs and unit tests were not considered or ranked highly in contribution to TD. Team members, across testing and development team, agreed that postponed fixes to meet deadlines was a bad practice, and contributed the most to TD. Interestingly, developers thought code without automated tests was a major TD category, while testers thought code that was redundant/not refactored was TD. These priorities reflected both in the relative ranking of TD criteria, and the contribution on the 5-point-scale.

Table 11: TD metrics, as prioritized by SMART Inc. team members

Dimension	**Metrics**	**Rank**	**Weight**	
Maintainability	Code that is released with errors in it	I	w_7	0.16

Maintainability	Outdated code - test script	IV	w_6	0.14
Quality testing	Defects found and not fixed	II	w_3	0.16
Quality testing	Code that is not tested, and not covered by automated tests	III	w_4	0.16
Quality testing	Code that is not tested, and not covered by manual tests	V	w_5	0.14
Process compliance	Code that has no associated unit tests	VI	w_1	0.13
Process compliance	Poor test case design	VII	w_2	0.11

Comparing with the interviews from different teams, which indicated an estimate of 10%-15% of TD in the projects, the survey participants estimated a large range, from 10%-90%, of code base has some degree of TD. The distribution of the responses' estimates is presented in Figure 19. On average, up to 44% of code base has TD.

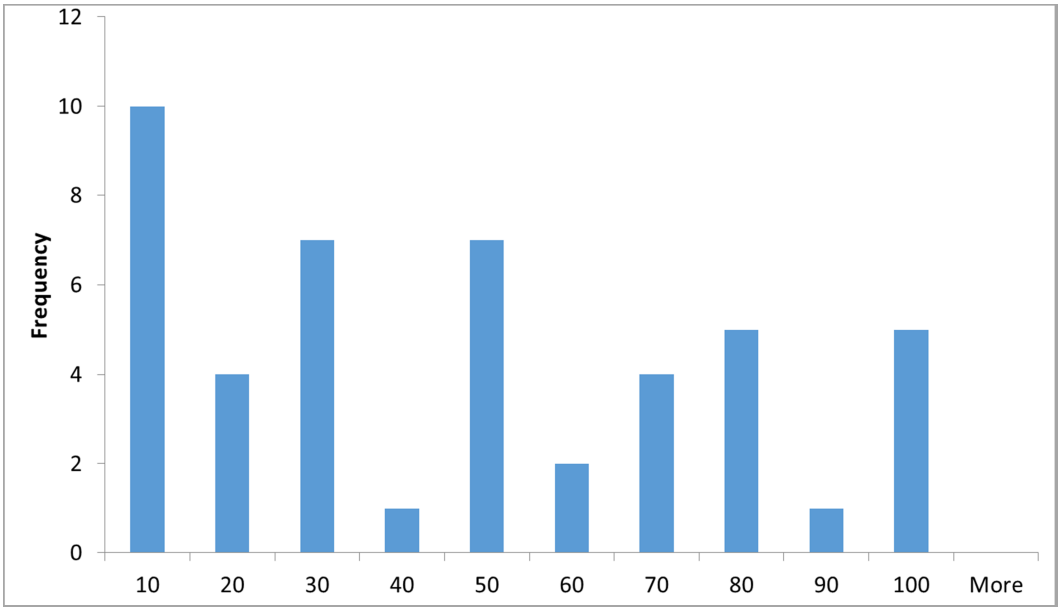

Figure 19: Histogram of estimates of percentage of TD in current projects[5]

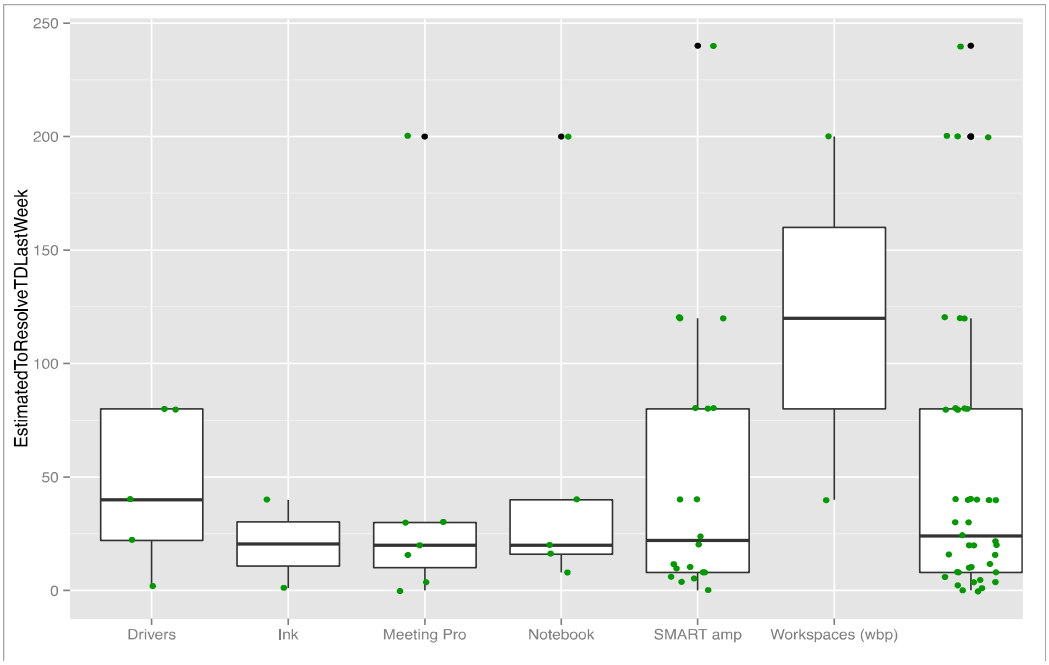

Figure 20: Estimated time spent working on non-optimal code [46]

In one question, we asked participants to estimate how much effort it would require to completely remove the TD they have observed from their project. The manual

[5] Boxplots created using R (http://www.r-project.org/) and Deducer (http://www.deducer.org/) software

evaluation estimated this amount of TD would require about 48.8 person-hours to eliminate, in the current projects. This figure was close to our tool-support estimates (in section *g* below). However, without continuous monitoring trend data, and the estimate provided was averaged from a small sample size, the estimation process would require further validation to confirm accuracy.

TD did affect productivity at SMART Inc. Participants estimated an increase in productivity, in some cases more than 70%, averaging around 30%, if TD was eliminated. The estimated increase in productivity (the value of TD reduction activity) is presented in Figure 21.

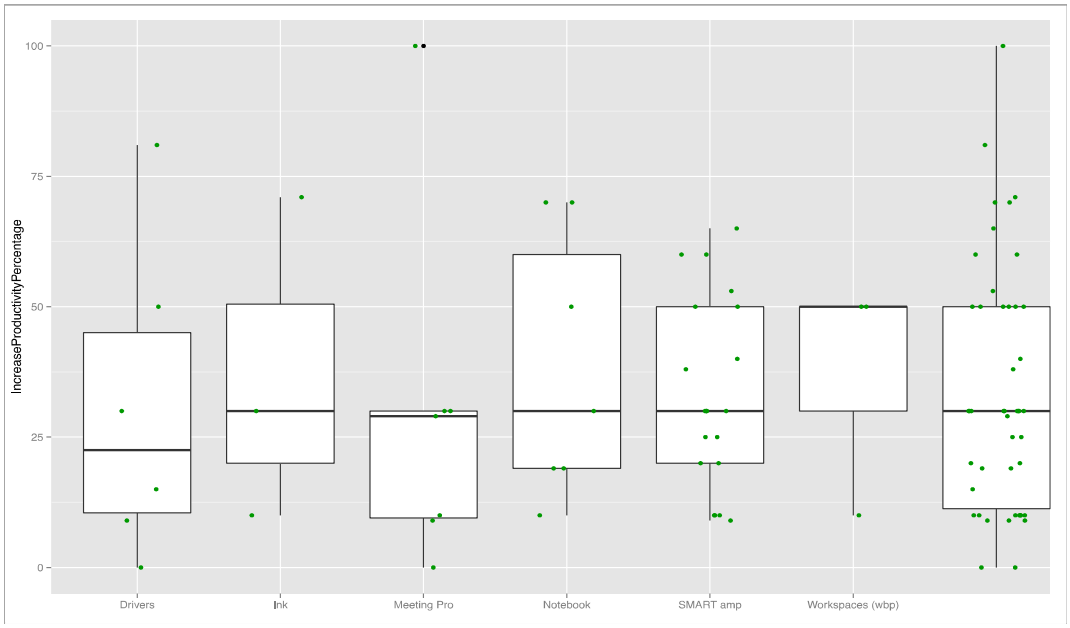

Figure 21: Projected increase in productivity (in %) if TD is eliminated [46]

This result was in-line with what we wanted to achieve in objective 1 (O1). While there was a high degree of adoption of Agile-Scrum practices at SMART, coupled with many tools provided for development and defect tracking and management, TD was largely overlooked. Team members acknowledged the existence of TD, yet provided varied estimates in term of what contributed to debt and how much there was for each

type of debt. There were no unanimous methods or common tools or plugins to visualize, monitor, and manage TD. Due to the (short) duration of the collaboration, we were not able to confirm the TD trend (to be growing over time). However, preliminary results showed that legacy systems which are more complex, and have been developed over a long period of time, tend to have larger TD estimates and variations.

Furthermore, in the results of the survey, there were some outliers (special) cases. Developers or testers estimated TD to be over 70% of code base, and would require more than 120 person-hours in a week of 40 person-hour of work to completely reduce debt (i.e. it requires more than one team members working full time on debt-ridden code). While these cases did not contribute to (and was not included in) our statistical analysis, they revealed key feedbacks for the team and the methods. Specifically, TD hindered productivity, and potentially made team members dissatisfied. There was more to TD than just quantifiable metrics. These consequences potentially lead to loss of trust, loss of morale, unstable environment and future unforeseen system downtime, etc.

The team members pointed out the lack of acknowledgement and visibility of TD in product planning and user stories management (in Scrum). Internal analysis had bugs trend, but did not include TD trend. These comments, while being outliers, signified a potential issue, and if followed up and presented appropriately, could trigger interests from management. Furthermore, they were also consistent with the goals of the plugin and the methodology provided in this research. This partially addressed the first part of our second study objective (O2): if TD is not appropriately managed, it can lead to non-optimal code and poor code practices.

2.2.7. Explorative recommendations

The second part of O2 is validated with the internal recommendations by team members, as collected from interviews and surveys, included here in Table 12. The top

three recommendations coincided with *Step 1* to *Step 4* of our proposed methodology. Team members were aware of TD. They converged in different aspects that contribute to TD. Furthermore, they aspired to define, track, and manage TD as part of the continual Agile-process of development. Notably, team members estimated a 30%-70% in productivity increase if TD is completely removed from the projects that they are working on. Below is the list of recommendations, grouped by their content.

Table 12: Internal recommendations by team members

Recommendations	Responses	Percentage
Plan for TD reduction into sprints and releases	9	18%
Allocate time and budget for TD reduction activity	15	29%
Increase TD visibility by monitoring and trend analysis	13	25%
Hire additional developers and testers to specialize in TD	3	6%
Do nothing; the team is managing TD well.	11	22%

Due to the time limitation of the collaboration, couple with the lack of an immediate milestone for release from the project examined, we were not able to generate the trade-off solutions. However, as an evaluation and validation for our TD monitoring steps, the case study at SMART provided valuable insights. In the following section, we discussed how the methodology could be applied in SMART Inc. context, and whether our recommendations had common-ground with the internal processes.

Based on the organization's feedback, the focus was on implementation of new key features, while starting a culture of incorporating TD reduction activity into sprint planning and execution. 25% of survey participants suggested defining TD in coding and testing practices, with 18% further proposed monitoring and viewing technical debt trends and data (corresponding to step 1 to step 3 of our methodology). 29% of

participants recommended TD to be planned into release activity, with one specifically called for not compromising TD activity for releasing new features (corresponding to what we aim to achieve in *Step 4 & Step 5*).

Following step 1 of our proposed workflow, we consolidated the different aspects of TD, and their relative importance to team members. GQM determined the following key metrics of TD to be included in Table 13. Using automated code analysis, and tracking issue tools available at SMART Inc., we collect the fitness score of the Whiteboard platform project (Core) as followed:

Table 13: TD metrics definition, weight and fitness score for SMART Core project

Dimension	Metrics	Weight		Fitness score
Process compliance	Code that has no associated unit tests	w_1	0.13	0.25
Process compliance	Poor test case design	w_2	0.11	0.22
Quality testing	Defects found and not fixed	w_3	0.16	0.01
Quality testing	Code that is not tested, and not covered by automated tests	w_4	0.16	0.20
Quality testing	Code that is not tested, and not covered by manual tests	w_5	0.14	0.01
Maintainability	Outdated code - test script	w_6	0.14	0.22
Maintainability	Code that is released with errors	w_7	0.16	0.09

In step 2 and 3, we analyze the source code available on August 2014 for the fitness score of each metric in each dimension, and aggregate them into one TD measure. *Figure 22* is the screenshot of the analysis of TD breakdown in the whiteboard project source

code snapshot as of August 2014. The estimated effort required to remove of TD in the example core (wbp) project was 40.24 person-days (for the team of 10-12 members).

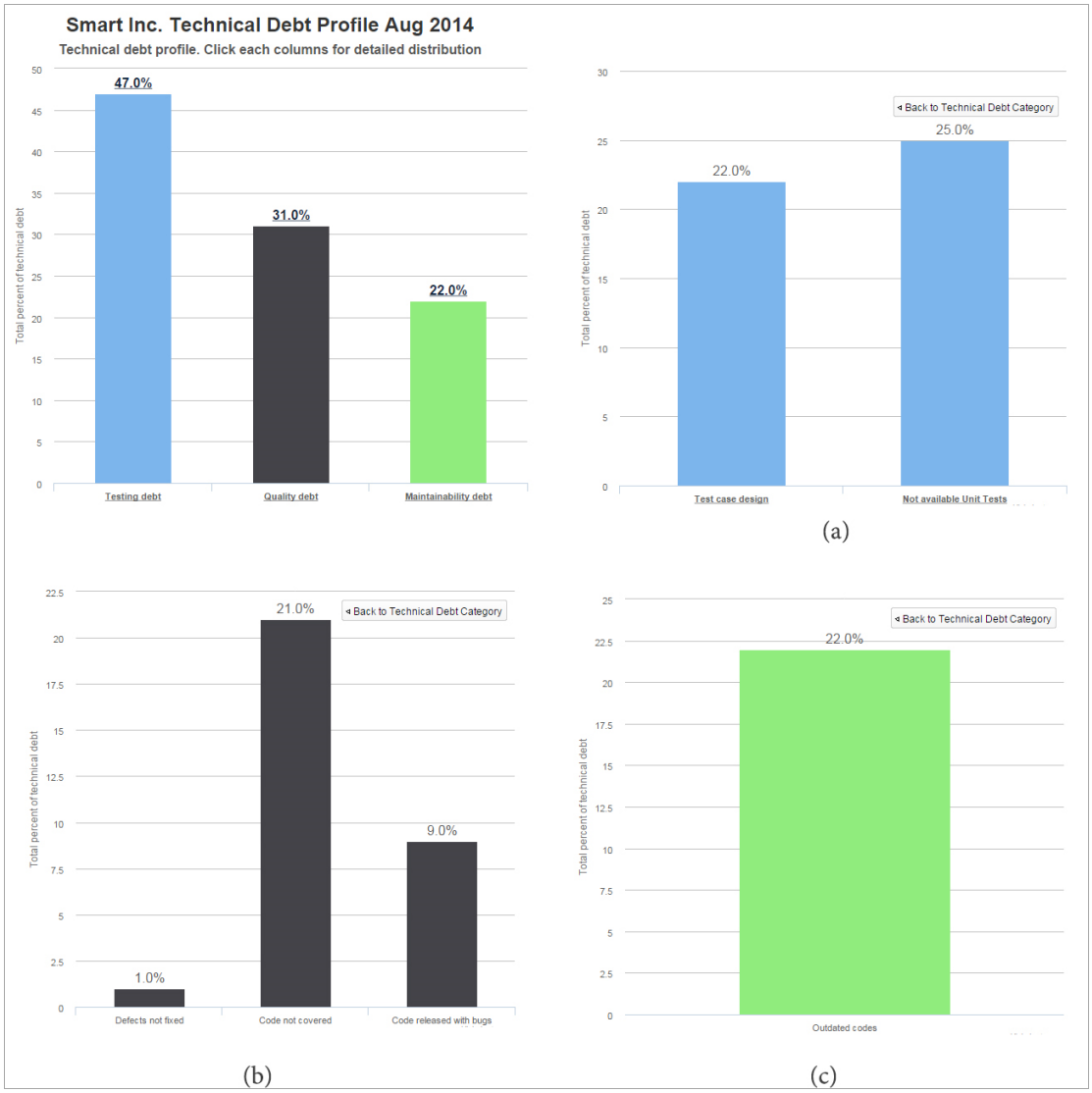

Figure 22: Technical debt breakdown at Smart in August 2014

(a) Testing debt (b) Quality debt (c) Maintenance debt

A presentation was made to stakeholders at SMART, presenting the findings summarized above, to both testing team and management team. The early response was positive. However, further studies would be required to validate the long term effect of this methodology, and establish more connection with when-to-release decisions.

In summary, through this case study and collaboration, we had some motivation on the real-world challenges of TD, and managing TD, even in an organization that adopted and followed Agile methods closely. The plugin tool was useful in identifying, measuring, and visualizing TD that were accumulated in one of the projects. The responses from human experts indicated interests and potential commitment to optimization and process improvement to be made.

2.3. Case Study 3: ReleasePlanner™ 2.0

In this case study, we examined a proprietary product from a start-up company. The case was of interest because, with the smaller team size and shorter release cycles, management of TD could potentially make high impact on the release date. Being the main platform where this plugin was designed upon, the project itself could provide seamless integrated data into the evaluation of W2RP-TD methodology. Lastly, as the project had accumulated TD that was both acknowledged by project managers and tracked by issues tracking tools, the optimization could then be validated, both numerically by effort allocation and in reality of impact to release time.

2.3.1. Study objective

Context: Given a proprietary product, with small but agile teams and short release cycles, and a product that needed to be released in a fix time frame

Objective 1 (O1): By integrating the TD plugin with existing issues management and tracking tools, we could visualize and monitor accumulated TD that had impacted development more effectively.

Objective 2 (O2): We applied the methodology and process directly to the project. We then collaborated with the project team to help them with when-to-release decisions. The results from this effort would validate that if TD is managed appropriately (implicitly by following our proposed methodology) on an organizational level, the impact to release

decisions (both what to release and when to release) could be optimized quantitatively. The combination of the proposed workflow and data analysis available through the plugin would ensure the release to be completed on time with desired set of functionality.

2.3.2. Organization background

Expert Decisions Inc. (EDI) [http://www.expertdecisions.com/] is a specialist player in the growing market of IT solution providers. The company has developed invaluable intellectual property and tools, capturing the release planning algorithms and philosophy. Research and innovation is done in strong interaction with the laboratory of Software Engineering Decision Support (SEDS) at the University of Calgary. In a systematic literature survey on strategic release planning models, SEDS was attributed a leading role world-wide in this area of research [93]. While being in market for more than 10 years, the team remains lean and agile. There are product management stream and development stream of work.

Furthermore, being a research and innovation-oriented organization, EDI has been supportive of the research and implementation done in this thesis. The research in the field of product management are adopted and designed into the new features of ReleasePlanner™, the company's flagship product.

2.3.3. Product background

Leveraging on the innovative researches as mentioned above, EDI offered a wide range of products in the strategic and operational release planning solution space, such as VeryBestChoice™, an intelligent analytic surveying tool, ReleasePlanner™, a decision support system in Product and Release management, and lastly, RASORP™, a scheduling and staffing component. The products offered by EDI have been applied in many academia and industrial projects, with customer profiles across industries and organizational levels.

In this case study, we focused on the main product offering of EDI, namely ReleasePlanner™ 2.0. ReleasePlanner™ is a proprietary web application that supports decision-makers on what-to-release decisions. The previous release of the product, capped at release 1.6 had been in production for more than 10 years, serving multiple clients in different domains [82]. As the legacy code base had been in production for such a long time, there was demand for a new major release. This demand stemmed from the aging software with high accumulated TD and complexity in change and maintenance, and the lack of support for modern platform such as mobile or tablet.

The new major release, Release 2.0, improved the usability and aesthetic of the interface, while implementing the latest breakthrough in researches developed in the SEDS group. The main competitive advantage of ReleasePlanner™2.0 was seen in the following capabilities:

- Offering optimized and diversified planning alternatives
- Consideration of quality criteria as part of the planning
- Integration of operational and strategic planning
- Comprehensive support of re-planning (both operational and strategic)
- Risk analysis capabilities
- Optimized what-if scenario playing capabilities
- Broad stakeholder involvement allowing customized prioritization based on specific criteria
- Ability to accommodate all types of logical feature dependencies [77]
- Integration into issue tracking and monitoring applications, such as Atlassian's JIRA, Microsoft's Team Foundation Service, etc.

2.3.4. Current state of development

The ReleasePlanner™ project had been selected as a case study for the thesis for the following characteristics, representative of Agile (Scrum-based) development in a small, flexible and nimble organization:

- **Team size:** as the product went through different (iterative) development life cycles, the team size varied. However, the team remained adaptable, with size ranging from 2 to 5 developers, a project manager, and 1-2 dedicated testers.
- **Project type and implementation methodology:** the project was a web based application. While the technology was proprietary and patented, the components designed and implemented remained modular and innovative. The team used iterative development cycles, with short releases of major functionality in releases, using ReleasePlanner™ for planning of the features implemented, and JIRA for issues and use case story tracking.
- **Availability of human experts:** As mentioned above, with the collaborative quality between EDI and SEDS, the human experts from the team, namely product manager, project manager, and lead developer were often available to evaluate our results and translate them into their actual decision-making process.

The project started in mid-2013, and was projected to go through 18 months of development. The functionality requirements of the system are presented in Appendix D. As the project followed iterative development cycles, many deliverables had been made. Table 14 outlined the progress of implementation of the desired functionality.

Table 14: Major deliverable of ReleasePlanner™2.0 (Present and Projected)

Work Package	**Main Activity**	**Delivery Date**	**Delivery status**
WP0	Solution Design	March-2013	Delivered

Work Package	Main Activity	Delivery Date	Delivery status
WP1.1	Baseline implementation (converting legacy functionality to new code base)	Jun-2013	Delivered
WP1.2	Implement interactive what-if analysis	Sep-2013	Delivered
WP1.3	Dashboard implementation	Dec-2013	Delivered
WP2.1	Implement advanced dependencies	May-2014	Delivered
WP2.2	Implement prioritization	Aug-2014	Delayed
WP3.1	JIRA integration – Import	May-2014	Delivered
WP3.2	JIRA integration – Export	Aug-2014	Delayed
WP4.1	UAT-Performance testing	Sep-2014	Delayed
WP4.2	Release version 2.0 to users	Jan-2015	Projected

As illustrated from Table 14, many modules had been delivered in the ReleasePlanner™ 2.0 project. However, many modules were either postponed or delayed. Technical debt quickly built up due to two major root causes: (i) the lack of documentation due to over-confident of the lead developer and (ii) accumulated testing and defect debts due to lack of balance between implementing new features and refactoring existing technical debt. Furthermore, the lead developer moved on to other projects, causing a gap in knowledge of implementation. The code base was tightly

coupled with new framework and technology that were not familiar with junior developers and interns.

This potentially risked the delivery of collaboration with business partners of EDI. TD was tracked starting from July 2014, where the delays started to happen, to get the project back on track. The analysis of technical debt and remediation recommendations are presented below.

2.3.5. Data analysis

Starting from the GQM definition process in our workflow (step 1), the team agreed on the following TD criteria and metrics in *Table 15*. Furthermore, using code and issue tracking analysis, in the month of September, the fitness score of the above metrics were assigned in *Table 15* that follows. How we calculated each fitness score is available in Appendix D, section C.

Table 15: TD metrics, weight and fitness score for ReleasePlanner™ in 09-2014

Dimension	Metrics	Weight	Value	Fitness
Process compliance	Not available documentation in artefact	$w_{d1\text{-}1}$	0.02	0.41
	Not available documentation in unit tests (TDD)	$w_{d1\text{-}2}$	0.05	0.20
	The percentage of non-documented interfaces.	$w_{d1\text{-}3}$	0.03	0.09
Quality testing	Number of defects found and not fixed at evaluation	$w_{d2\text{-}1}$	0.25	0.87

Dimension	Metrics	Weight	Value	Fitness
Quality testing	The percentage of code that is not tested, and not covered by automated tests	w_{d2-2}	0.2	0.41
	The potential threat to security due to outdated, broken code	w_{d2-3}	0.05	0.10
Maintainability	The interdependency between components	w_{d3-1}	0.08	0.80
	The degree of platform or tool dependency	w_{d3-2}	0.08	0.80
	The modular degree of code, the percentage of duplicate code	w_{d3-3}	0.04	0.50
Complexity	Degree of legacy system and components available	w_{d4-1}	0.12	0.39
	Current system completed code size	w_{d4-2}	0.08	0.42

The technical debt profile is presented in the below *Figure 24*. The data of this breakdown is available as part of Appendix D. Due to brevity of the thesis, we would only list here the major root causes and discuss the potential remediation for the project. We presented here the period between July 2014 and November 2014, with focus at the September 2014 snapshot. Technical debt was consistent at 5.7% range of total

development effort. However, as there were more code and implementation being added, TDAF was on an upward trend, with quality testing comprising the most to overall TD.

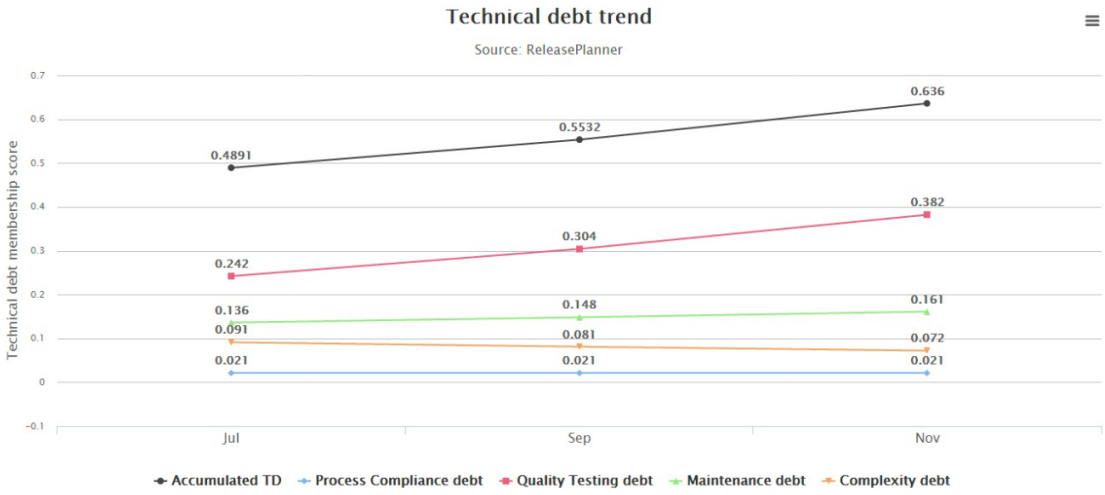

Figure 23: RP2.0 technical debt trend during Jul-Nov 2014

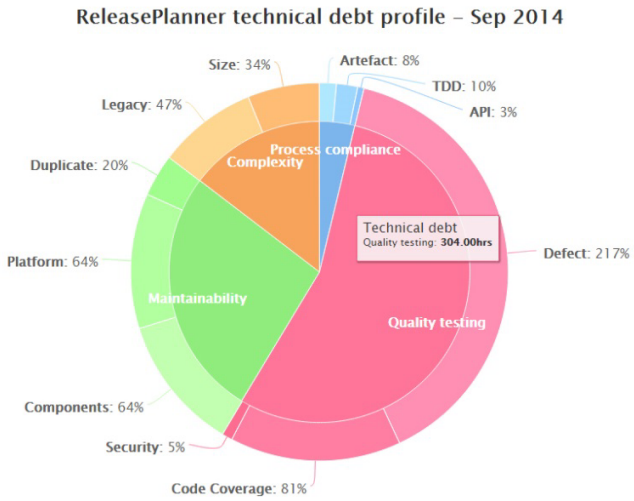

Figure 24: Technical debt Profile of ReleasePlanner™ 2.0

The root causes of these technical debts were analyzed.

- Documentation debt: many of the required functionality had no to little documentation. As the development cycle followed both Scrum methodology [79] and Test-driven-development (TDD) [65], which were characterized by the

inherent lack of documentation, using the "code as documentation". The lead developer often challenged the project management team with the trade-off "Would you rather I spend time coding functionality, or writing documentations?" The technical debt in documentation further accumulated as he moved on to a different project, leaving code hardly readable by new developers who came on-board later on in the project.

- Design debt: There was a lack of discussion in architectural design for the solution. Often, the lead developer decided which approach to use and implemented them directly into the main frame, without either design or documentation. Further complicating the architectural design was that many independent frameworks were applied, for the "convenience of coding", in exchange for understandability and documentation of the code.
- Testing debt: While testing was identified as a major activity of the project, defects often remained unfixed for an extended period of time. The project had good unit testing code coverage (Appendix D, *Figure 34*). There had been two rounds of UAT testing. However, as implementation trailed behind, many modules were blocked and therefore not available for testing.
- Implementation debt: Remaining defects remained in the system, logged in issue tracking, revealed two unhealthy trends for defect debt: (i) defects remained in the system for a long time before getting fixed and (ii) once selected for fix, developers took a long time to fix each defect. This was an indication that the code has become harder to maintain as the debt-ridden code hindered the fixing of defects.

2.3.6. Recommendations

Given the original plan for the next release to be in January 2015, TD had to be carefully planned into integration. The project needed to be stabilized over the next

release, balancing between offering new functionality with removing outstanding issues and defects. Remediation recommendations and trade-off solutions (step 4-5) are presented in the following screenshots.

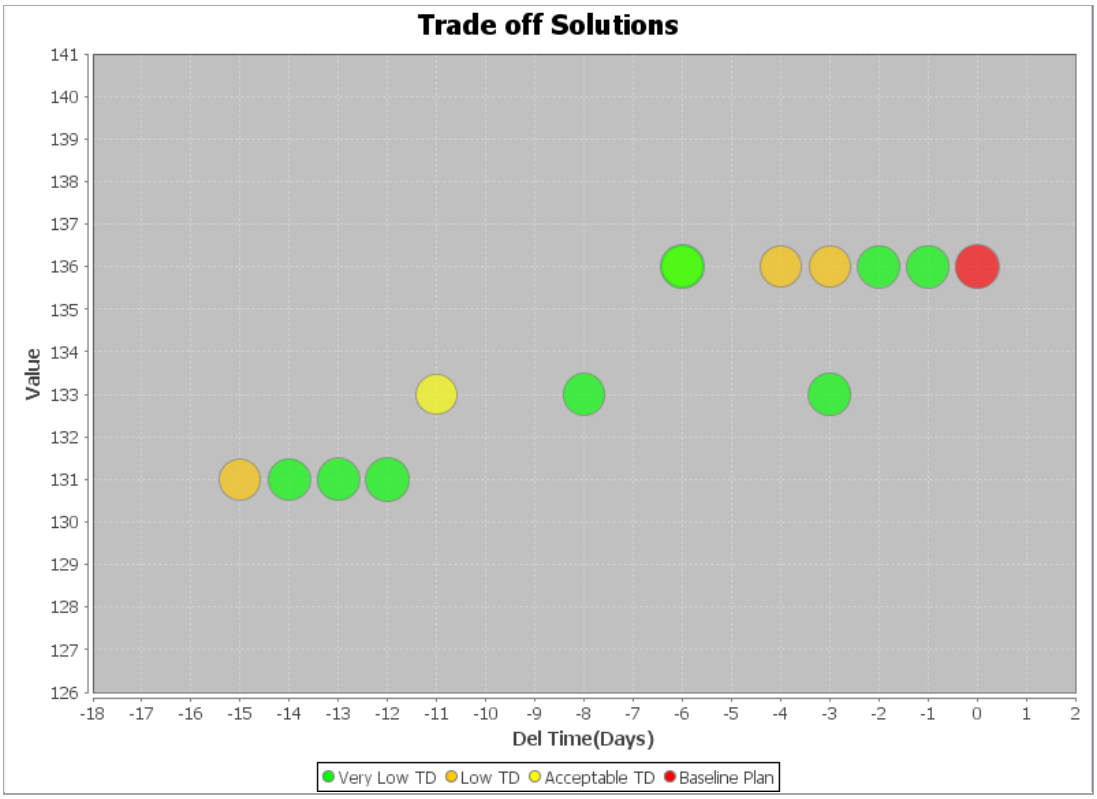

Figure 25: Trade-off solutions as suggested by the W2RP-TD plugin for Jan release

We can see that there was a diverse set of solutions. A discussion with the development team determined that stabilizing the project, addressing technical debt should take precedence at this stage. Therefore, we only considered the 'Very low TD' solutions, in this case. From the what-if playing scenarios, the product was ready for release as planned, with full set of originally offered featured, if 2 person-days of working effort was invested into reducing TD (which will improve TD by 2.4%). We presented here a comparison on the Dashboard among baseline solution, incurring debt to release (1 day) early solution, and addressing debt by extending (2 days) solution.

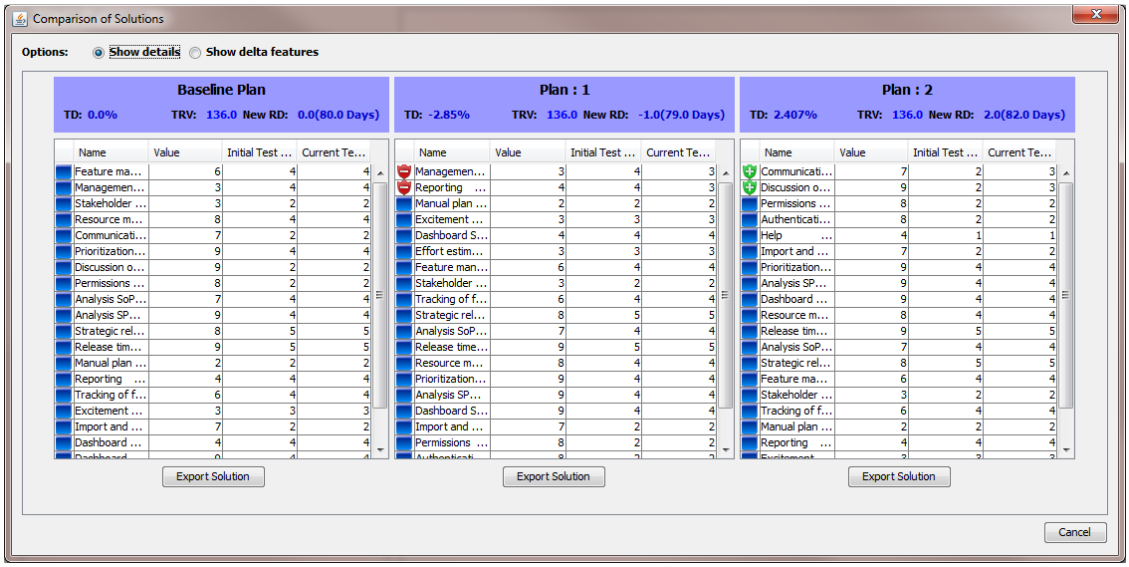

Figure 26: Solutions proposed by W2RP-TD plugin tools

The final selected solution (+2 days, -2.4% TD) was planned and exported back into ReleasePlanner™ and JIRA for execution (step 6). Figure below presents the operational view of the implementation, as of November 2014. This plan had been reviewed by stakeholders of the project. At the point of writing, the plan and the project fix rate was back on track.

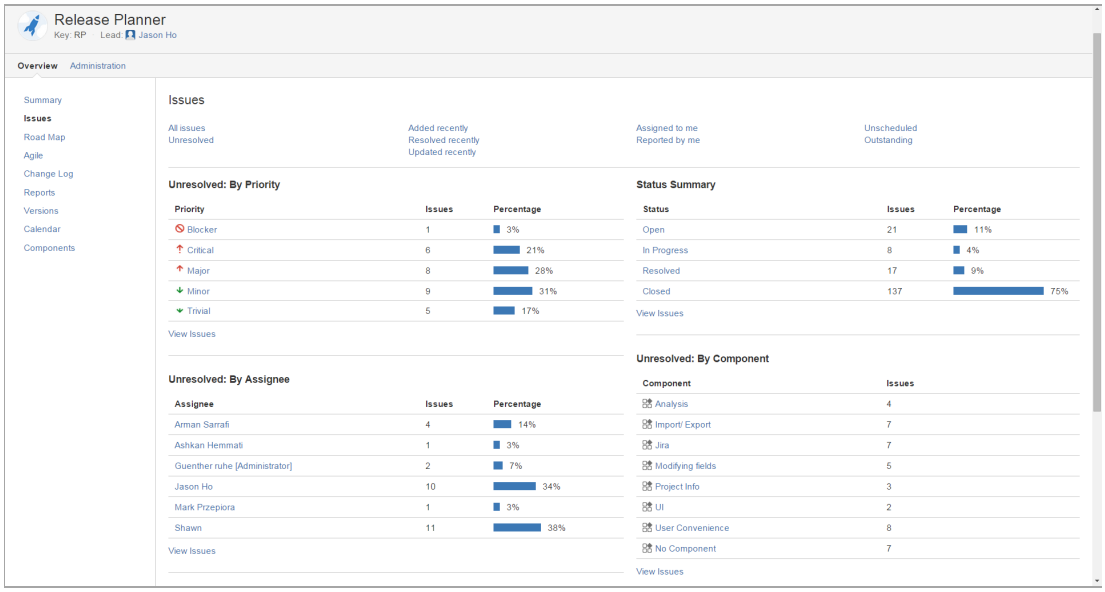

Figure 27: Exported work plan into JIRA for implementation

3. Threats to validity

Technical debt (TD) is a relatively new research field [68]. There has not been enough models and methodology for comparison and validation. The results in these case studies are from multiple publications and collaborations in the course of three years. We are aware of the potential threats to validity that may impact the outcome and usefulness of this research in real-time.

We will discuss the general threats of validity to the research and methodology, while citing specific cases where each case study may have been impacted. We then follow each of these identified threats with the execution measures taken to minimize them. These threats also present potential for future researches in the same directions to improve the collected result.

3.1. Conclusion validity

The methodology proposed in this thesis is still in the early stage of development. Limited industrial data, coupled with the challenges in obtaining such data in real-world, complex projects, may potentially hinder the conclusions findings in the results presented above. Technical debt (TD) measurement is a relatively new research field, which definitions not yet clearly defined [42]. Therefore, our findings in this research will require further promotion and adoption within the community to be useful.

With the three case studies, even with similar sets of GQM, TD consisted of very different types and sizes of elements. Especially with OSS projects, the data collected was partial. The comparison with existing tools in industry, as expected, revealed different aspects of TD. However, the trend of growth of TD appeared to be consistent over the period of two years.

We addressed this threat through extensive literature review, and repeating this methodology for multiple projects and research collaborations. The concept presented in this thesis was partially presented at the Sixth International Workshop on Technical Debt Management (MTD) [49] and was well received by the audience.

3.2. Construct validity

One key threat to construct validity of this approach is the modeling and measurement of technical debt. TD quantification is complex, especially in the case of integrated testing scenarios of large systems, as proven from literature [2]. The application of the tool requires extensive data about effort estimate and cost drivers in project projects. This can be addressed with further experiments, similar to tuning in COCOMO II model. The model requires extensive data collection and tuning in a period of time before it can be applied broadly in industrial context.

In *Assumption 1*, Chapter Three:. Section 1.1, we assume that features decision can be made independently, or can be modeled using advanced constraints formulation. While this is a potential limitation for the model due to the nature of complex inter-dependency between features, in our experience [35], most features can be modeled this way. Even when the modeling is not complete, the initial assessment and trade-off proved helpful to the decision making process [XCIX]. In case study 2 and 3, features are implemented in this way, and assigned into sprints accordingly. For case study 1, OSS projects have issues that are raised by the community and tracked effectively for the period of minimum 2 years.

Lastly, we inspire our technical debt estimate on COCOMO II model [9]. The TD formulation and estimation is inspired by empirical based measurement such as COCOMO II, which is validated and experienced from real-world data. While intuitively there are many similarities between effort estimation for software implementation and TD

reduction, the cost drivers need to be further investigated, and tuned accordingly to the TD context. Correlation between TD and effort cost drivers need to be established. To address this threat, we conducted multiple studies in different domains and organizations. More study of this kind is planned for future work as part of the follow-up of this research.

3.3. Internal validity

In this research, TD is modeled in four dimensions, with multiple criteria in each dimension. In reality, one dimension may have an influence on the other dimensions. For instance, poor documentation could lead to a lot of defects in the system, causing accumulated quality testing. We minimize this dependency by using a set of metrics that are not overlapping, as specified in the GQM selection criteria, as explained in III. B.1. Similar to the COCOMO II cost drivers; these metrics require further validation in industrial case study to be applied. Such studies are planned in the follow up of this work.

In OSS projects, we measure TD based on (limited) available public data, and establishing the relationship with when-to-release decisions. There may be unforeseen (and not available) information that affected the releases that we are not aware of. To help eliminate this threat, we compared our results with popular TD measurement techniques such as SQALE or CAST. We further investigated using any public available secondary information sources on the release quality, such as that of forum posting or Github code reviews and issues reports.

In real-world case studies such as case 2 and 3, there are overhead of effort transfer, besides implementation and testing, such as task-switching, operational activities, etc. when-to-release decisions maybe impacted by factors than just technical debt and feature implementation. However, these efforts can be fine-tuned and adjusted from the operational profile of the project, further increase the accuracy of the recommendations.

Internal validity is maintained by having the author keeping track of all the projects during the process. All collaborations are monitored closely with a set of pre-defined metrics. Raw data are saved for future references and retrospective correction should discrepancies are discovered.

3.4. External validity

External validity is managed by analyzing and filtering data in relation to each other. There are external partners and products, which are challenging to control. However, data that are relevant to the case and to technical debt, as outlined by related work and known methods in the field, are collected and analyzed. Data that were not available was not extrapolated, but rather extracted from similar legacy products where applicable, or from similar industry standards.

As there are only limited number of case studies and technical debt data available in these case studies, it is hard to claim external validity for the method. It is however possible to validate the effectiveness of this methodology through extensive literature review and survey, which has been conducted during the completion of this thesis. As presented in Chapter Six:, we evaluate different organizations in different industry, at different stages of development. More research in this direction are planned to expand and further prove the effectiveness of the technique and/or the plugin tool.

Chapter Seven: CONCLUSIONS

1. Overview

In this thesis, we examined the current practices and methodologies available to effectively track, manage and reduce technical debt (TD) in the context of release planning. Based on this background and related works, we proposed a methodology and workflow to improve TD management and integrate these activities into release cycles in the form of trade-off solutions.

In summary, we answered the following research questions, which are the focus and central to this thesis:

RQ1: *How well do existing models, methods, or support tools efficiently and effectively model and manage technical debt?*

RQ1.1: What are the current methods and metrics in measuring and quantifying technical debt?

In Chapter Two:, Section 3, we explored the different related work in technical debt (TD). We concluded that, although TD is a relatively new research field, there have been many different proposals, definitions, and approaches to quantify and manage it effectively.

RQ1.2: Do these metrics and methods identify above efficiently and effectively support the when-to-release decisions for (software) products?

However, through both II.C and II.D, we pointed out that there is a lack of consistency in TD measurement and management in both academic researches and industry practices. Furthermore, most existing TD measurement methodologies focus exclusively on measuring the principle (the amount of accumulated debt) without much estimation on interest, cost, and value of TD activities.

RQ2: *How can we evaluate, classify and formulate (both qualitatively and quantitatively) technical debt in relations to release decisions?*

RQ2.1: How can we measure and estimate technical debt in relations to effort required to reduce them?

In Chapter Three:, Section 2, we formulate TD quantitatively using GQM method, coupled with known techniques in software estimation (i.e. COCOMO II) to quantify TD in term of effort required to reduce it, in relation to the total effort of features and projects implementation.

RQ2.2: How can we manage technical debt in relations to when-to-release decisions?

In Chapter Three:, we tie all the different aspects of a release plan together, relating the business pressure of when-to-release decisions with the accumulated TD in a project or product. We then propose a method to balance TD with W2RP decision using optimization based on local effort re-allocation and global non-dominated trade-off solution between release date, business value and TD, in Chapter Four:.

RQ3: *How can we provide decision support for creating and selecting the release plans that maximize readiness (business values and quality) while minimizing the negative impacts of technical debt?*

RQ3.1: Is it feasible for release decisions to be supported by a prototype tool implemented based on the above proposed methodology?

In Chapter Five:, Section 1 and 2, we outlined the motivation and the implementation behind the prototype tool that implemented the methodology proposed in Chapter Four:. The tool is designed as a plugin, utilizing existing industrial platforms such as ReleasePlanner and JIRA. The technical solution blueprint follows MVC framework, and utilizing the latest in GUI and Java implementation.

RQ3.2: What are the characteristics required for the design of such prototype tool?

The requirements of the tool are outlined coupled with a walkthrough in Chapter Five:, Section 3. The tool is a direct implementation and translation of the methods and workflow proposed in Chapter Four:.

RQ3.3: How do we evaluate the methodology and validate the prototype tool in real-life, complex projects?

Chapter Six: presented all validations that have been done, at the point of writing, in different organizations, product lines, and projects. This diversity serves as a form of investigation of the usefulness (and limitation) of the method in real-world, real-time and complex software projects. Recommendations were made and retrospective evaluation is conducted for this validation. The results and findings collected are still at early stages, yet, the method proves helpful in (i) creating visibility of technical debt within the project teams and (ii) supporting informed, analytic-based decision making process in when-to-release planning.

2. Contributions

As reviewed in section A, we have answered the research questions (RQ) that we have previously set out to achieve. By answering these questions in the research methodology and validation approach, the thesis has made the following contributions, as previously outlined in Chapter One:, Section 5:

1. A methodology to estimate technical debt (TD) in relation to work effort required to reduce it to the (ideal state of) zero debt in the system. In measuring TD as percentage of the total work effort required to implement features and releases, we address previous criticisms of TD management practice including that of 'perfect system' or 'systems with debt larger than value' [25]. The method is simple enough to minimize the cost of human

experts' involvement, yet comprehensive enough to measure TD in all its dimensions, as defined internally and based on context using GQM method.

2. A workflow procedure to evaluate all possible trade-off solutions between the when-to-release software dates and the permitted level of TD accumulated in the system. By quantifying and optimizing TD reduction and business values of features, W2RP-TD empowers product managers to make informed decisions about whether and how much debt they are willing to incur into the system in order to meet delivery deadlines. The methodology and its implementation serve as a starting point for organizations to adopt a systematic process of TD management and release planning. The method, thanks to the flexibility of GQM monitoring and COCOMO II fine tuning of adjustment factors, can then be further implemented and adjusted to fit organizations to be helpful long term.

3. A prototype plugin tool that interacts and works with data from existing industrial tools for release planning (i.e. ReleasePlanner™) and issues tracking and management (e.g. JIRA, etc.). The prototype utilizes the evaluation of TD, coupled with the optimization capability of these existing tools, to generate and visualize the set of trade-off solutions for product managers. As of this writing there is no such tool existing in the market or research area of TD and W2RP. The visualization aspects of TD trend, TD breakdown, and direct trade-off between release date, business values and TD empower business managers to make informed technical decisions.

4. Lastly, we conducted preliminary empirical evaluations of how useful the above contributions are in real-world products and organizations. The results are still early and to be further developed. At the point of writing, the information has been deemed useful by software product teams. Several publications and the analyses available in the previous chapters demonstrated this contribution in diverse organizations and products. Product managers interviewed indicate interests in the significance of TD management information, management technique, and release planning proposed.

3. Future Work

As stated from empirical study, technical debt (TD) is a new research field, and the methodology and plug-in tool in this thesis are in its early stage of development. Further development, validation, and promotion of the techniques need to be carried out to evolve the usefulness of the thesis.

In this research, business values and technical debt are modeled based on estimated efforts, which are uncertain in nature and require fine-tuning from extensive real-world data to be meaningful in practice. More experiments and adjustments of TD dimension factors and adjustment factors are planned for future work. The results provided in this thesis are illustrative examples of the direction such similar investigations can follow. We initiated the validation and data collection from different organizational size and product releases cycles. This can be replicated to future explorative and data collection work.

Technical debt is an industry-driven concept [24]. Researches in this field have been driven both by industry and academia. However, benchmarks and standard definitions have yet to be reached [42]. This thesis, coupled with related works in this space, aspire to contribute to become part of that missing standards. In order to achieve this, the method needs to be applied to a wider range of both industrial and academic projects. Data needs to be collected and analyzed and made accessible to both communities. Thus far, the organizations we approached have embraced the concept of TD and how it can help with release process. This thesis can serve as a guideline for future empirical work and adoption into industry standards.

Furthermore, the prototype is integrated as a plugin into existing industrial issue tracking tools such as JIRA and release planning tools to collect and analyze testing and requirements data from these tools in real time. Follow up work in term of solution design and integration are planned to bring the methodology to industry practice. The

method has received positive feedback so far from international workshop and conferences [49], we intend to further develop the prototype and get it fully deployed and integrated into organizations.

Finally, as the methodology in this thesis for decision support in when-to-release decisions have a predictive component. In order to validate the 'quality' of the recommendation, further follow up and tracking is needed. As indicated by the financial debt metaphor, as argued by Cantor [18], TD management is similar to insurance policy, where investment of now will prevent future issues. Long research data collection cycles are required to confirm the usefulness of the method. TD affects multiple facets of product development, not just coding and code quality. Therefore, the methodology needs to be applied on organizational level to procure the most tangible benefits. This has been proposed to multiple of our collaborators in the case studies mentioned, and those who seek to improve and manage TD in their organizations.

4. Closing statements

Technical debt is a powerful metaphor for researchers and practitioners to understand the trade-off between business pressures to deliver on-time and potential consequences in maintainability and changeability of the product in the future. In this research, we investigated a multi-faceted formulation of TD, and quantify them directly in effort trade-off with new features implementation and release dates. Through tool support and multiple empirical validation studies, we aspire to empower product managers and project managers to make better decision in managing their product releases, in direct consideration with technical debt. The methodology and the plugin created as part of this thesis, though in early stages, have proven to be useful in the case studies that we have conducted during the duration of this study.

References

[1] E. Allman, "Managing technical debt.", in *Communications of the ACM,* vol. 55, no. 5, 2012, pp 50-55.

[2] A. Asthana, and J. Olivieri, "Quantifying software reliability and readiness", in *Communications Quality and Reliability*, *2009. CQR 2009. IEEE International Workshop Technical Committee on*, IEEE, 2009, pp. 1-6.

[3] A. J. Bagnall, et al., "The next release problem", in *Information and Software Technology*, vol. 43, no. 14, 2001, pp. 883-890.

[4] V. R. Basili, G. Caldiera, D. Rombach , and R. van Solingen, "The Goal Question Metric Approach", in *Encyclopedia of Software Engineering*, Marciniak J (ed.), vol. 1, pp. 578-83, Wiley 2001.

[5] P. Bhawnani, G. Ruhe. "ReleasePlanner-Planning new Releases for Software Maintenance and Evolution", in *ICSM (Industrial and Tool Volume)*, 2005.

[6] K. Bittner, I. Spence, "Managing Iterative Software Development Projects", Addison-Wesley, 2007.

[7] B. Boehm, "Software risk management: principles and practices", in *Software, IEEE*, vol. 8, no. 1, 1991, pp. 32-41.

[8] B. Boehm, "Value-based software engineering: Overview and agenda", in *Value-Based Software Engineering*, Springer Berlin Heidelberg, 2006, pp. 3-14.

[9] B.W. Boehm, et al., "Software Cost Estimation with COCOMO II", Prentice Hall, 2000.

[10] B. Boehm, and J. Bhuta, "Balancing opportunities and risks in component-based software development", in *Software, IEEE* vol. 25, no. 6, 2008, pp. 56-63.

[11] B. Boehm, V. R. Basili, "Defect Reduction Top 10 List", in *Computer*, vol. 34, no. 1, 2001, pp. 135-137.

[12] R. Brettschneider, "Is your software ready for release?", in *Software, IEEE* vol. 6, no. 4, 1989, pp. 100.

[27] A. W. Brown, M. Delbaere, P. Eeles, S. Johnston, and R. Weaver, "Realizing service-oriented solutions with the IBM rational software development platform", in *IBM systems journal*, vol. 44, no. 4, 2005, pp.727-pp.752.

[14] N. Brown, *et al,* "Managing technical debt in software-reliant systems", in *Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM*, November-2010, pp. 47-52.

[15] S. Buchholz, and D. Sisk, "Technical debt reversal, Lowering the IT debt ceiling" in *Tech Trends 2014: Inspiring Disruption*, vol. 2. 24., Deloitte University Press, 2014, pp-66-75.

[16] F. Buschmann, "To pay or not to pay technical debt", in *Software, IEEE*, vol. 28, no. 6, 2011, pp. 29-31.

[17] J. E. Burge, et al., "Rationale-based software engineering", Springer, 2008.

[18] M. Cantor, "Calculating and improving ROI in software and system programs." in *Communications of the ACM*, vol. 54, no. 9, 2011, pp. 121-130.

[19] D. N. Card "Practical software measurement.", in *Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society*, 2003, pp. 738-739.

[20] P. Carlshamre, "Release Planning in Market-Driven Software Product Development: Provoking an Understanding", in *Requirements Engineering*, vol. 7, 2002, pp-139-151.

[21] R. N. Charette, "Software engineering risk analysis and management, Intertext Publications, 1989.

[22] S. Cleveland, and T. J. Ellis., "Orchestrating End-User Perspectives in the Software Release Process: An Integrated Release Management Framework."

[23] P. Conroy, "Technical Debt: Where Are the Shareholders' Interests?" in *IEEE Software,* vol. 29, no. 6, 2012.

[24] W. Cunningham, "The WyCash portfolio management system", in *SIGPLAN OOPS Mess.*, vol. 4, no. 2, Dec 1992, pp. 29-30.

[25] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the size, cost, and types of Technical Debt.", in *Managing Technical Debt (MTD), Third International Workshop on, IEEE*, 2012, pp. 49-53.

[26] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II.", in *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, 2002, pp. 182-197.

[27] P. Deemer, G. Benefield, C. Larman, and B. Vodde, "The scrum primer". in *Scrum Primer is an in-depth introduction to the theory and practice of Scrum, albeit primarily from a software development perspective*, available at: http://assets.scrumtraininginstitute.com/downloads/1/scrumprimer121.pdf 1285931497, 2010.

[28] M. Denne, and J. Cleland-Huang, "The incremental funding method: Data-driven software development", in *Software, IEEE*, vol. 21, no. 3, 2004, pp. 39-47.

[29] S. Devnani-Chulani, "Modeling software defect introduction", in *Proc. California Software Symposium'97*, 1998.

[30] C. Ebert, S. Brinkkemper, “Software product management – An industry evaluation” in *The Journal of Systems and Software*, 2014 , http://dx.doi.org/10.1016/j.jss.2013.12.042

[31] C. Ebert, M. Bundschuh, R. Dumke, and A. Schmietendorf, "Making Metrics a Success—The Business Perspective," in *Best Practices in Software Measurement: How to use metrics to improve project and process performance*, 2005, pp. 9-34.

[32] R. J. Eisenberg, "A threshold based approach to technical debt", in *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 2, 2012, pp. 1-6.

[33] N.A. Ernst, "On the role of requirements in understanding and managing technical debt", in *2012 Third International Workshop on Managing Technical Debt (MTD), IEEE*, 2012.

[34] D. Feitelson, E. Frachtenberg, and K. Beck, "Development and Deployment at Facebook.", 2013.

[35] M. Felderer, A. Beer, J. Ho, and G. Ruhe, "Industrial evaluation of the impact of quality-driven release planning", in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 62. ACM, 2014.

[36] J. Fisher, D. Koning, A. P. Ludwigsen. "Utilizing Atlassian JIRA for Large-Scale Software Development Management", in *14th International Conference on Accelerator & Large Experimental Physics Control Systems (ICALEPCS)*, 2013.

[37] Forrester Research Inc., "Increase Business Flexibility By Embracing Future Trends", May 2012.

[38] O. Gaudin, "Evaluate your technical debt with Sonar," in *Sonar*, 2009.

[39] Z. Gery. (2013, Jan 23). *Technical Debt: Strategic vs Non-Strategic* [Online Document]. Available: http://www.codeproject.com/Articles/597822/TechnicalplusDebt-3aplusInplusanplusAgileplusWorld , viewed 04/08/2013

[40] A. L. Goel, "Software error detection model with applications", in *Journal of Systems and Software*, vol. 1, 1980, pp. 243-249.

[41] D. Greer, and G. Ruhe, "Software release planning: an evolutionary and iterative approach", in *Information and Software Technology*, vol. 46, no. 4, 2004, pp. 243-253.

[42] I. Griffith, D. Reimanis, I. Clemente, Z. Codabux, A. Deo, and B. Williams, "The Correspondence between Software Quality Models and Technical Debt Estimation Approaches", in *Sixth International Workshop MTD on Management Technical Debt at ICSME,* 2014.

[43] R. Gulezian, "Reformulating and calibrating COCOMO", in *Journal of Systems and Software*, vol. 16, no. 3, 1991, pp. 235-242.

[44] S. Gueorguiev, M. Harman, and G. Antoniol, "Software project planning for robustness and completion time in the presence of uncertainty using multi objective

search based soft-ware engineering", *Paper presented at the 11th Annual Conference on Genetic and Evolution-ary Computation, Montréal, Québec, Canada*, 8-12 July, 2009.

[45] M. Harman, "The current state and future of search based software engineering", in *Future of Software Engineering, IEEE Computer Society*, 2007, pp. 342-357..

[46] J. Ho, G. Ruhe, S. M. Didar-Al-Alam, R. Karim, and Z. Shakeri, "Technical debt decisions in testing with cost/benefit optimization", a case study conducted at Smart Inc., 2014.

[47] T. Hegazy, "Optimization of resource allocation and leveling using genetic algorithms", in *Journal of construction engineering and management*, vol. 125, no. 3, 1999, pp. 167-175.

[48] J. Ho, S. Shahnewaz and G. Ruhe, "A Prototype Tool Supporting When-to-release Decisions in *Iterative Development", in 2nd International Workshop on Release Engineering (RELENG)*, 2014.

[49] J. Ho and G. Ruhe, "When-to-release Planning in consideration of Technical Debt", in *Sixth International Workshop MTD on Management Technical Debt at ICSME,* 2014.

[50] H. Huang, D. Ho, J. Ren, and L. F. Capretz, "Improving the COCOMO model using a neuro-fuzzy approach", in *Applied Soft Computing*, vol. 7, no. 1, 2007, pp. 29-40.

[51] C.Y. Huang, S.Y. Kuo, and M. R. Lyu, "An assessment of testing-effort dependent software reliability growth models", in *Reliability, IEEE Transactions on*, vol. 56, no. 2, 2007, pp. 198-211.

[52] H. Johannes, V. Leppänen, and S. Hyrynsalmi, "Technical Debt and the Effect of Agile Software Development Practices on It–An Industry Practitioner Survey", in *Sixth International Workshop MTD on Management Technical Debt at ICSME,* 2014.

[53] H.W. Jung, "Optimizing value and cost in requirements analysis", in *IEEE Software*, vol. 15, no. 4, 1998, pp. 74-78.

[54] B. A. Kitchenham, "Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods", in *ACM SIGSOFT Software Engineering Notes,* vol. 21, no. 1, 1996, pp. 11-14.

[55] T. Klinger, et al., "An enterprise perspective on technical debt", in *Proceedings of the 2nd Workshop on Managing Technical Debt*, ACM, 2011.

[56] T. Kremmel, J. Kubalík, and S. Biffl, "Software project portfolio optimization with advanced multiobjective evolutionary algorithms", in *Applied Soft Computing* vol. 11, no. 1, 2011, pp. 1416-1426.

[57] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: from metaphor to theory and practice", in *IEEE Software*, vol. 29, no. 6, 2012, pp.18-21.

[58] P. Kruchten, R. L. Nord, I. Ozkaya, and D. Falessi, "Technical debt: towards a crisper definition", in *report on the 4th International Workshop on Managing Technical Debt, SIGSOFT Software Engineering Notes,* vol. 38, no. 5, 2013, pp. 51-54.

[59] D. Leffingwell, "Scaling Software Agility", Pearson Education, 2007.

[60] J.L. Letouzey, T. Coq, "The SQALE analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code" in *Proceedings of the 2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, IEEE Computer Society, 2010.

[61] X. Li, "A risk-based approach for software release time determination with delay costs consideration", in *Transactions on Software Engineering*, 2010.

[62] E. Lim, N. Taksande, C. Seaman, "A balancing act: what software practitioners have to say about technical debt" in *Software, IEEE*, vol. 29.6, 2012, pp. 22-27.

[63] L. Juuso, "Portfolio decision analysis for robust project selection and resource allocation", Teknillinen korkeakoulu, 2008.

[64] M. Lang, and C. Barry, "A survey of multimedia and web development techniques and methodology usage", 2001.

[65] L. Madeyski, “Test-Driven Development”, Springer, 2010.

[66] A. Malik, V. Pandey, and A. Kaushik, "An Analysis of Fuzzy Approaches for COCOMO II", in *International Journal of Intelligent Systems and Applications (IJISA)*, vol. 5, no. 5, 2013, pp.68.

[67] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," in *IBM Journal of Research and Development*, vol. 56, no. 5, 2012, pp. 9:1-9:12.

[68] S. McConnell, "Managing Technical Debt“, Construx Software Builders, Inc, in *Software Development Best Practices*, 2008.

[69] J. McElroy, G. Ruhe, “When-to-release decisions for features with time-dependent value functions”, in *Requirements Engineering Journal*, vol. 15, 2010, pp. 337-358.

[70] C. Morris A., J. Eliasberg, T.H. Ho, "New product development: The performance and time-to-market tradeoff." in *Management Science,* vol. 42.2, 1996, pp. 173-186.

[71] C. J. Neill, Phillip A. Laplante, "Paying down design debt with strategic refactoring.“, in *Computer*, vol. 39, no. 12 , 2006, pp. 131-134.

[72] K. Okumoto, and A. L. Goel. "Optimum release time for software systems based on reliability and cost criteria", in *Journal of Systems and Software 1*, 1980, pp.315-318.

[73] D. Parnas, "Software aging", in *Proceedings of the 16th international conference on Software engineering*, *IEEE Computer Society Press*, 1994, pp. 279-287.

[74] C. Patel, and M. Ramachandran. "Agile maturity model (AMM): A Software Process Improvement framework for agile software development practices", in *International Journal of Software Engineering*, IJSE 2, no. 1, 2009, pp.3- pp.28.

[75] R. S. Pressman, “Software Engineering, A Practitioner’s Approach”, 3rd Edition, McGraw Hill, New York, 1992, p.559.

[76] M. Poppendieck, and T. Poppendieck, “Lean software development: An Agile toolkit”, Addison Wesley, Boston, Massachusetts, USA, 2003.

[77] M. Przepiora, “A Hybrid Release Planning Method for Accommodating Advanced Feature Dependencies”, Master thesis, 2012.

[78] T. S. Quah, "Estimating software readiness using predictive models", in *Information Sciences,* vol. 179, no. 4, 2009, pp. 430-445.

[79] L. Rising, N. S. Janoff, "The Scrum software development process for small teams", in *IEEE Software*, vol. 17.4, 2000, pp.26-32.

[80] J. Rhodes, CM First Group, “Technical Debt –Why it Matters to Capacity Managers”, white paper, 09-2013.

[81] R. Y. Rubinstein, and D. P. Kroese, “Simulation and the Monte Carlo method”, vol. 707, John Wiley & Sons, 2011.

[82] G. Ruhe, “Product release planning: methods, tools and applications”, CRC Press, 2011.

[83] G. Ruhe, “Software engineering decision support–a new paradigm for learning software organizations”, in *Advances in Learning Software Organizations*, Springer Berlin Heidelberg, 2003, pp. 104-113.

[84] P. Runeson, and M. Höst, "Guidelines for conducting and reporting case study research in software engineering", in *Empirical software engineering*, vol. 14, no. 2, 2009, pp. 131-164.

[85] A.P. Sage, “Decision support systems engineering”, Wiley-Interscience, 1991.

[86] A. S. Sayyad, and H. Ammar, "Pareto-optimal search-based software engineering (POSBSE): A literature survey", in *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*, IEEE, 2013, pp.21-27.

[87] Seaman, Carolyn B. "Qualitative methods in empirical studies of software engineering", in *Software Engineering, IEEE Transactions on*, vol. 25, no. 4, 1999, pp.557-572.

[88] C. Seaman, *et al.,* "Using technical debt data in decision making: Potential decision approaches", in *Managing Technical Debt (MTD), 2012 Third International Workshop on, IEEE*, 2012, pp. 45-48.

[89] C. Seaman, Y. Guo, "Measuring and monitoring technical debt", in *Advances in Computers,* vol. 82, 2011, pp. 25-46.

[90] S. Shahnewaz, and G. Ruhe, "RELREA - An Analytical Approach for Evaluating Release Readiness", in *International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2014.

[91] J.P. Shim, et al., "Past, present, and future of decision support technology", in *Decision support systems* vol. 33.2, 2002, pp. 111-126.

[92] C. Sterling, "Managing Software Debt: Building for Inevitable Change", Addison-Wesley Professional, 2010.

[93] Svahnberg, Mikael, et al. "A systematic review on strategic release planning models", in *Information and software technology*, vol. 52, no. 3, 2010, pp. 237-248.

[94] N. Taksande, "Empirical study on technical debt as viewed by software practitioners", master thesis, University of Maryland, Baltimore County, 2012.

[95] D. A. V. Veldhuizen, and Gary B. Lamont, "Multi-objective evolutionary algorithms: Analyzing the state-of-the-art", in *Evolutionary computation*, vol 8, no.2, 2000, pp.125-147.

[96] K.E. Wiegers, "Software Requirements", Microsoft Press, 2003.

[97] K. Wiklund, et al., "Technical Debt in Test Automation", in *Proceeding in Software Testing, Verification and Validation (ICST)*, IEEE, 2012.

[98] A. Williams, DevelopersArena, "GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz", Online document, 07-2012, http://developersarena.com/web/2012/07/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreessen-horowitz/ , viewed 11-2014.

[99] C. Wohlin, et al., “Experimentation in software engineering”, Springer, 2012.

[100] A. Wood, “Software reliability growth models: assumptions vs. reality” in *Proceedings from the Eighth International Symposium on Software Reliability Engineering*, IEEE. 1997.

[101] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking", in *Systems, Man and Cybernetics, IEEE Transactions on,* vol. 18, no. 1, 1988, pp. 183-190.

[102] G. Zorn-Pauli, B. Paech, B., T. Beck, H. Karey, and G. Ruhe, “Analyzing an Industrial Strategic Release Planning Process–A Case Study at Roche Diagnostics. Requirements Engineering: Foundation for Software Quality”, Springer, 2013.

[103] Smart website: http://www.smarttech.com/, viewed 08/02/2014

[104] EDI website: http://www.expertdecisions.com/, viewed 09/17/2014

Appendix A – Illustrative examples of trade-off solutions generation

1. Approach

In this appendix, we illustrate with an example of how the optimization in Chapter Four:, Section 2 is being executed. To recall, the pseudo-code of the trade-off operation is as followed:

Line	Function Generate Solutions (F_0:=Baseline plan, F_N:= Features pool, ΔRD:=change in release duration)
1	Define Baseline F_0 ← (TRV(F_0), TD(F_0), RD(F_0))
2	Define: S := Solution pool, S* := Trade-off solution
3	Initialize solution pool S ← [ ]
4	While not TerminateCondition() do
5	Vary RDi: ΔRD ≤ Max(ΔRD)
6	For each RDi, generate plan F_i so that ∑E(f(i)) + E_{TD} ≤ Cap(RD_i)
7	NewPlan F_i ← F_i (TRV(F_i),TD(F_i), RD_i)
8	For all F* in S*:
9	If F_i is superior to F*: Eliminate F* from S*
10	If F* is superior to Fi: Eliminate Fi from S
11	Merge (S, S*)
12	Return trade-off solutions S*

2. Numerical example

With reference to the ReleasePlanner™ project, whose product background and functionality is outlined in Chapter Six:, Section 2.3 and Appendix D.

From this list of functionality, the baseline release plan F_0 has been determined for RD_0 to be January 31st 2015. For line 1 of the pseudo code, F_0 is created and stored as reference in the tool. In line 2 & 3 of the code, the solution pool is initially emptied. So is the trade-off solution pool.

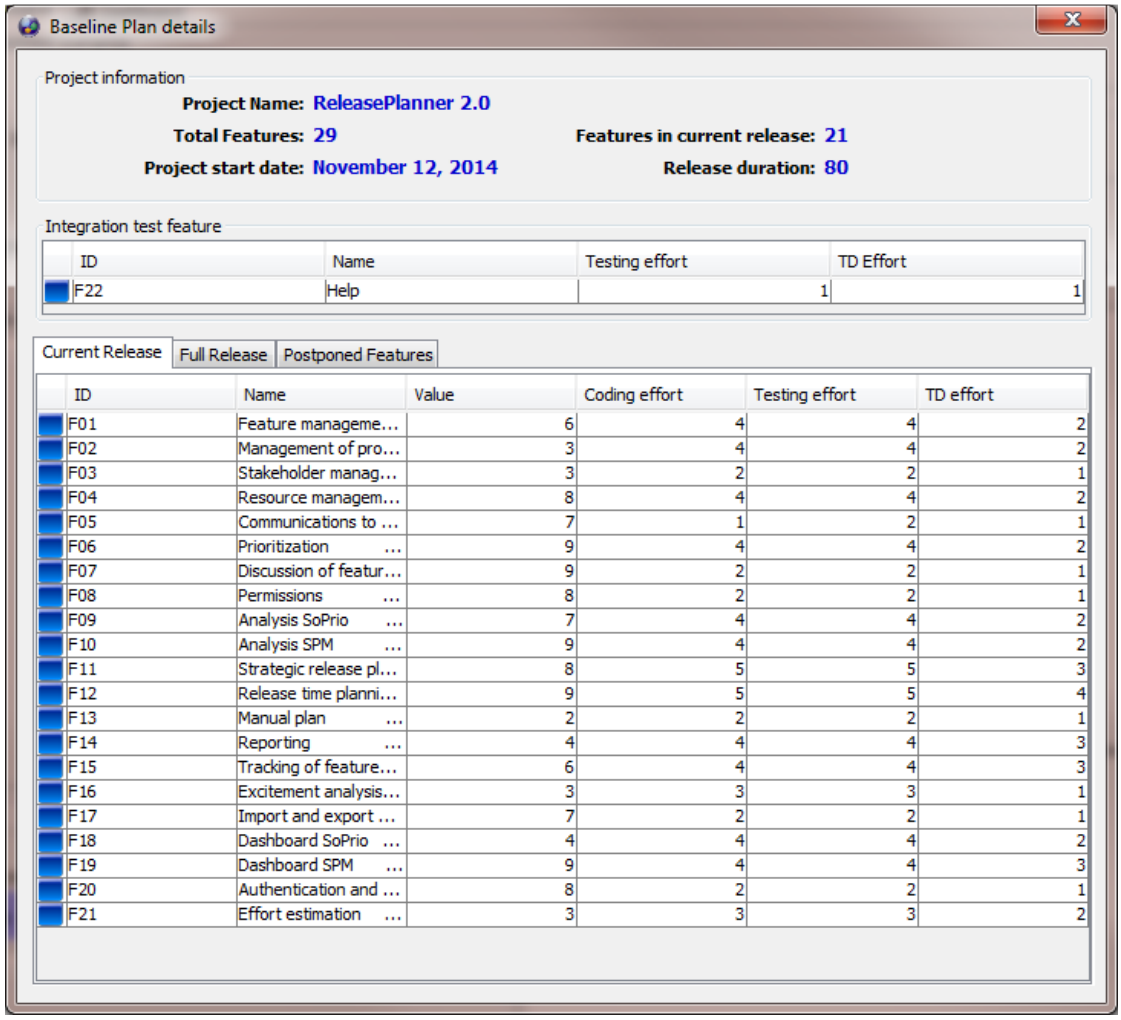

Figure 28: Baseline release plan for ReleasePlanner™ 2.0

We started the loop on line 4 when the users input all their settings. For this example, in the setting parameters, we set ⊠RD = -10 days.

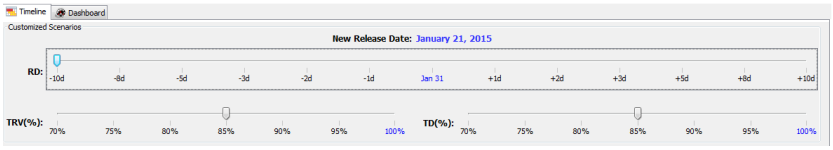

Figure 29: Setting parameters for the effort re-allocation

This corresponds to line 5 of the pseudo-code. We will vary the release date within the range of 5 days. In other words, we loop through RD_i with RD_i = -1, -2, -3 … -9, -10.

For line 6 and line 7, we look at generating all possible solutions for each RD. Let's examine ⊠RD = -8. This translated to 64 person-hours per team member on the team. This time should be re-allocated between implementation, testing, or fixing effort for each feature. Each feature, as presented from the baseline, has prioritized business value, implementation effort, and testing effort.

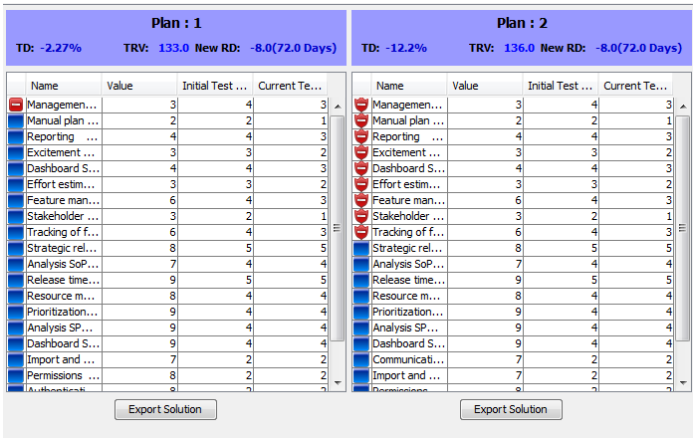

Figure 30: Comparing between two trade-off solutions

To generate the plan, the solution ranked features based on their value and importance. For the effort that needed to free up, the solution either eliminate (less important but high effort) features, like *Plan 1* on the left. It can also incur TD by reduce testing and fixing effort, like in *Plan 2*. All these plans, as long as they are eligible are stored in solution pool S.

In line 8 to 10, we compare the solutions pair-wise in value between TRV and TD. Only solutions superior in TRV, or TD (or both) remains. Solutions that are inferior is

eliminated. In line 11, we compared all the solution in S with existing in S*. If there are solution in S* with the same RD, but less features, or more TD, we eliminate such solution from S*. Reversely, if there are solution in S* that has the same TRV and TD, but can be accomplished in faster RD, the solution in S is eliminated. S* is then merged with solutions in S, and S is then emptied.

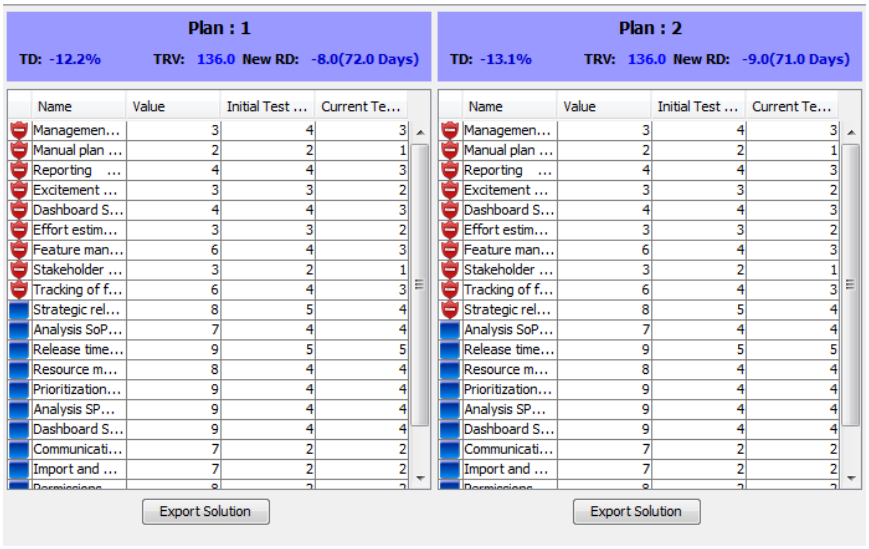

Figure 31: Trade-off plans in different RDs

This operation is looped until no more RD variation remained. The solutions in S* are returned and visualized.

3. Solutions recommendation example

Combining all the trade-off solutions available from the trade-off pool, we have 4 solutions available:

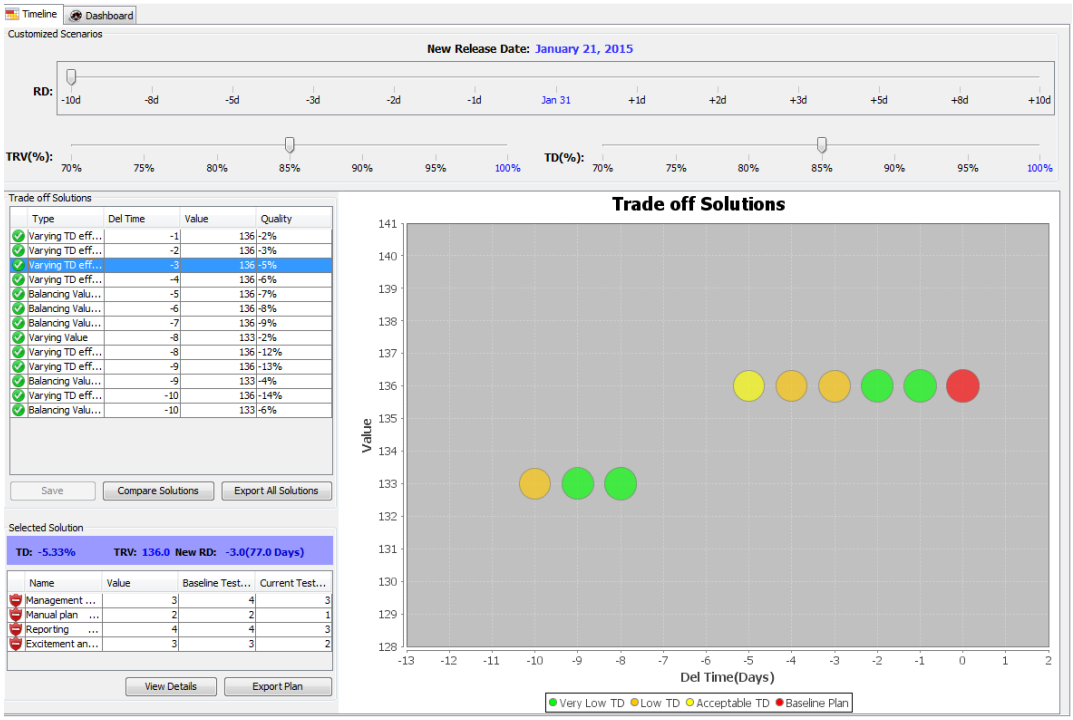

Figure 32: Trade-off solution, with RD = -3 being selected

The screen above show how the recommendations are being displayed, with the detailed of what TD to incur corresponding to this solution.

Appendix B – Interview questionnaires

Automated testing & technical debt (TD)

1. Objective of the study / interview:

The objective of this study is to understand and formulate technical debt (TD) at [organization] with specific focus on test automation, with potential recommendations for improvement in the effectiveness of test automation.

2. Introduction – to be explained to participants before interview starts:

As part of the initiative to implement better products in shorter time to market, [organization] has adopted organization-wise automated testing to maximize test coverage and products quality. However, with the business pressure of ever shorter release duration, there is a rising concern that larger code coverage alone will not result in less defects and less technical debt during the development process. We understand that, at [organization], coding and testing is highly automated, and supported by many different tools. With the pressure of having all projects having more than 80% test coverage, there is also concern of redundant test cases, and long run time of tests.

This particular study will examine existing practices of automation in testing coupled with manual bug testing. We would like to gather real-world data to determine how qualitative and quantitative optimizations can be designed to decrease automated testing run time, increase code quality, and decrease technical debt. Using this advanced data analytics, we hope to enable decision makers to make the best decisions of putting out the best products in the shortest timeframe, with high test coverage and full control of potential risks and opportunities entailed by technical debt.

3. **General guidelines:**

- All questions need to gear towards the objective mentioned above
- We need to understand the process at [organization], starting from the day-to-day operations of what these people do, keeping in mind that this is their perspective only and may not be a complete representation of [organization].
- Whenever possible, as quantitative questions such as time spent on certain task, or distribution and prioritization of tasks. This will reveal interesting aspects of how people consider technical debt and/or implementation in relation to each other.

I. Interview questions:

Questionnaire 1: target audiences: **Program/Product Manager**

We asked the global degree of representation of the interviewee from representative of [organization] to locally to our teams.

We need to let them know these questions are specific to the current project they are working on. We need to ask if they work on other projects/products as well.

Q. 1. Where do you get the ideas for new features requests from? How do you prioritize the features request? Is there revenue information, if not, what scale do you use for considering new features/change requests?

- Expected Data Collected: Prioritization process, potential revenue or 9-point-scale values points for features based on their desired levels (Very high to very low)
- How does it relate? Understanding the source of deficits in technical debt (TD): is there business pressure to deliver new features faster? How well informed are the product managers of the development process, and how much do they take into consideration the recommendations from development team when making decisions?

- Expected follow-up questions: In the event of conflicting priority, how do teams normally resolve such conflict in the Agile process?

Q. 2. How do you spend your work hours in your daily activity? Example: How many hours a week is dedicated to talking to stakeholders (end-users, project teams, development teams, etc.)?

- Expected Data Collected: Operational profile of Product manager, how they prioritize and interact within the teams in Smart
- How does it relate? Understand the process of decision making
- Expected follow-up questions: Have you ever experienced a time when you have to take your time off from your daily activity to address a client's concerns in Software, or something not working as expected?

Q. 3. How do you think software impact the sales of hardware at [organization]? In the event that a desired feature or functions are (or not) available in a hardware, do you have an estimate of how much revenue difference that decision would make?

- Expected Data Collected: A revenue figure, or a way to come up with this figure, at Smart. Ed said this information is not available, but we should ask anyways.
- How does it relate? Directly impact the trade-off between whether or not TD should be addressed at Smart.
- Expected follow-up questions: What would you consider as a catastrophic event to the product, how often is this event software-related?

Q. 4. Are you familiar with technical debt (TD) concept? Do you think development team should spend time improving existing software (versus keep building new features)? If you are to decide how much time the team should invest in this 'improvement' activity, how much would it be, 10%, 20%, etc. of development time?

- Expected Data Collected: A TD budget as understood and allowed by an MBA Program Manager. The potential cause for lack of TD management in Smart.
- How does it relate? The time constraint dev team has to improve their current code

Questionnaire 2: target audience: **QA manager / testers**

We asked the global degree of representation of the interviewee from representative of [organization] to locally to our teams.

We need to let them know these questions are specific to the current project they are working on. We need to ask if they work on other projects/products as well.

We asked the global degree of representation of the interviewee from representative of [organization] to locally to our teams.

We need to let them know these questions are specific to the current project they are working on. We need to ask if they work on other projects/products as well.

Q. 1. How do you spend your work hours?

a. Example: How many hours a week is dedicated to implementing new feature, fixing bugs, creating test suites, etc.?

b. What is the escalation process? If you have a concern/conflict of interest, how is this concern heard or resolve?

- Expected Data Collected: Process of time budget and conflict resolving at Smart.
- How does it relate? Indirectly reveals how Smart resolve and prioritize their internal-conflict in relation to quality management and new features implementation

c. What would you consider as a catastrophic event to the product, how often is this event Software related?

- Expected Data Collected: Operational profile of developers or how they prioritize and interact within the teams in Smart

- How does it relate? Understand the day to day work of developers
- Expected follow-up question: What are your pain points? What stop you from being as productive as you want to be? If there is one thing you can improve about work process in and out of your team, what would it be?

Q. 2. How do you manage recorded (fogbugz) issues?

a. Does repeated or long running tasks (ex: functional test run time) affect your productivity? If so, how does it affect you?

b. When an issue is discovered by you, do you work on it right away? How do you prioritize (blocker to trivial) defects?

c. How long do you need to investigate the root cause of such issue, (if at all)?

d. How do you evaluate the effort it would take for you to regression test and close issue?

e. Do you include developing UT test for the recorded issue in your estimate?

- Expected Data Collected: Hours spent on operational activity, in direct relation to testing and defects (one main part of debt)
- How does it relate? Directly translate to the level of debt in the system
- Expected follow-up questions: How much of your unit/functional testing is automated? Do you have ideas to improve it? What are they? How much time do you need to implement your ideas?

Q. 3. Do you spend time on non-features or not recorded issues such as designing additional automated tests, etc., may not be immediately needed, but will improve the project overall quality?

a. Is refactoring activities included as a Work item? Is it included in Sprint meeting for assignment?

b. If not, what would trigger a refactoring or improvement activity?

c. On the scale of 1-5 (very little – huge impact), how much will this impact overall quality of the code-base/project?

d. On the scale of 1-5 (ad-hoc – frequent), how often do you do this type of activity? Who initiates this? Ex: ad-hoc, management, etc.

Expected Data Collected: Whether or not the interviewee has ideas about Technical debt, and the degree of control they think they have over technical debt.

How does it relate? Directly impact the trade-off between whether or not TD should be addressed at Smart.

Expected follow-up questions: How critical are these activities to the code? If we don't do this now, do you think this will affect future versions of the Software? How long until something really bad happen because we never did these activities?

Q. 4. Are you aware of the code coverage objective at [organization]?

a. For example, for current project they are working on, it's 80%. What does test coverage of 80% mean to you?

b. Do you have different objectives (80%, etc.) for different types of testing? Different parts of the functionality?

c. How do you decide which should be tested automatically and which should not be?

- Expected Data Collected: A bottom-up guideline on what to test, what not to test, what to fix, what not to fix.
- How does it relate? Reveal the areas that can be improved in automated testing.

Expected follow-up questions: How strong do you think are these testing KPI is enforced? If other team meets this requirement, does it help with your own testing/coding? Do you think improvement to meet 80% score is realistic or superficial? Do you think it can be further improved above 80%? If so, how should we do it, in your opinion?

Questionnaire 3: target audience: **Development project manager/developers**

We asked the global degree of representation of the interviewee from representative of [organization] to locally to our teams.

We need to let them know these questions are specific to the current project they are working on. We need to ask if they work on other projects/products as well.

Q. 1. How do you spend your work hours?

a. Example: How many hours a week is dedicated to implementing new feature, fixing bugs, creating test suites, etc.?

b. What is the escalation process? If you have a concern/conflict of interest, how is this concern heard or resolve?

- Expected Data Collected: Process of time budget and conflict resolving at Smart.
- How does it relate? Indirectly reveals how Smart resolve and prioritize their internal-conflict in relation to quality management and new features implementation

c. What would you consider as a catastrophic event to the product, how often is this event Software related?

- Expected Data Collected: Operational profile of developers or how they prioritize and interact within the teams in Smart
- How does it relate? Understand the day to day work of developers
- Expected follow-up question: What are your pain points? What stop you from being as productive as you want to be? If there is one thing you can improve about work process in and out of your team, what would it be?

Q. 2. How do you manage recorded (fogbugz) issues?

a. Does repeated or long running tasks (ex: functional test run time) affect your productivity? If so, how does it affect you?

b. When an issue is discovered by testers and assigned to you (developers), do you work on it right away? How do you prioritize (blocker to trivial) defects?

c. How long do you need to investigate the root cause of such issue?

d. How do you estimate the effort it would need for you to fix an issue? How do you evaluate the effort it would take for you to regression test and close issue?

e. Do you include developing UT test for the recorded issue in your estimate?

- Expected Data Collected: Hours spent on operational activity, in direct relation to testing and defects (one main part of debt)
- How does it relate? Directly translate to the level of debt in the system
- Expected follow-up questions: How much of your unit/functional testing is automated? Do you have ideas to improve it? What are they? How much time do you need to implement your ideas?

Q. 3. Do you spend time on non-features or not recorded issues such as designing additional automated tests, etc., may not be immediately needed, but will improve the project overall quality?

a. Is refactoring activities included as a Work item? Is it included in Sprint meeting for assignment?

b. If not, what would trigger a refactoring or improvement activity?

c. On the scale of 1-5 (very little – huge impact), how much will this impact overall quality of the code-base/project?

d. On the scale of 1-5 (ad-hoc – frequent), how often do you do this type of activity? Who initiates this? Ex: ad-hoc, management, etc.

- Expected Data Collected: Whether or not the interviewee has ideas about Technical debt, and the degree of control they think they have over technical debt.

- How does it relate? Directly impact the trade-off between whether or not TD should be addressed at Smart.
- Expected follow-up questions: How critical are these activities to the code? If we don't do this now, do you think this will affect future versions of the Software? How long until something really bad happen because we never did these activities?

Q. 4. Are you aware of the code coverage objective at [organization]?

a. For example, for current project they are working on, it's 80%. What does test coverage of 80% mean to you?

b. Do you have different objectives (80%, etc.) for different types of testing? Different parts of the functionality?

c. How rigorously do you follow TDD? Is there a unit test or functionality test for everything you do?

- Expected Data Collected: A bottom-up guideline on what to test, what not to test, what to fix, what not to fix.
- How does it relate? Reveal the areas that can be improved in automated testing.
- Expected follow-up questions: How strong do you think are these testing KPI is enforced? If other team meets this requirement, does it help with your own testing/coding? Do you think improvement to meet a 80% score is realistic or superficial? Do you think it can be further improve above 80%? If so, how should we do it, in your opinion?

Appendix C – Human experts' technical debt survey

1. Introduction:

Thank you for agreeing to participate in this survey. This survey will take no more than 15 minutes of your time.

Technical debt (TD) is a metaphor used to refer to the consequences of shortcuts taken during the design, implementation, testing and documentation of software. In our organization, we haven't yet quantified TD and cannot accurately assess the potential risk that it poses. The results of this survey will enable us to better understand, control and manage TD related to testing.

2. Ethic approval form

The ethic approval form for this survey is enclosed at the end of the questionnaire.

3. Survey Questionnaire

Part A: Demographic information:

Question 1: Which team are you from?

A. Product Management

B. Testing

C. Development

D. Others. Please specify: _____

Question 2: What project are you currently working on?

Project name: _____

Question 3: How many years have you been in the project mentioned in Question 2?

A. Less than 1 year

B. 1 – 2 years

C. 2 – 5 years

D. More than 5 years

Only for development team or testing team:

Question 4: How many years have you been developing/testing software (in your career)?

A. Less than 1 year

B. 1 – 2 years

C. 2 – 5 years

D. More than 5 years

Part B: Technical debt questions:

Question 5: How much do you agree with the following statements?

☐ 1: Strongly disagree

☐ 2: Weakly agree

☐ 3: Neutral

☐ 4: Weakly agree

☐ 5: Strongly agree with these statements

- We improve testing by adding new test cases
- We improve testing by maintaining/refactoring existing test cases
- The test run time affects my productivity
- Technical debt affects my productivity
- Having knowledge of the current status of TD would be helpful
- We should address technical debt regularly, at least once a week
- We should address technical debt regularly, at least once each sprint
- We should address technical debt regularly, at least once every quarter

There are many different facets of TD for example documentation, architecture, and Coding. In this survey, we are primarily interested in the TD that occurs during

development and testing. The focus of the next question will be on different components of technical debt related to testing.

Think about the TD from testing in your current project. What aspect of technical debt in testing do you spend the most time working on? Please sort the following components of TD according to whether you spend a High, Medium or Low amount of time working on them:

☐ Poor test case design.

☐ Code that has no unit tests attached to them.

☐ Code that has no automated test cases assigned to them.

☐ Code that requires explorative manual testing, but no such manual testing assigned to them.

☐ Code that is tested and flagged as potential error, but not fixed.

☐ Test cases that failed but code is released anyway due to deadline and postponed till later fixes.

☐ Obsolete (or unused) code but no availability to remove or refactor it from the code base yet.

Question 7: Think of your current project. In the last week what percentage of the code that you worked with had some form of TD?

_____%

Question 8: As an estimate: How many hours would it take to resolve all the TD that you observed last week?

_____# of hours

Question 9: If we could reduce all TD in testing to zero, what percentage productivity (increase/decrease) would this you experience in your personal velocity? Please provide an estimate (where no change is 0%, values range ±100%):

_____ %

Question 10: How many person-days (in a month) should be dedicated to reducing TD in testing? Please provide an estimate (values range 0-30 days):

_____ # of person-days

Question 11: What are your suggestions on reducing testing TD in your current project?

_____

Thank you for your participation!

Appendix D – ReleasePlanner™ 2.0

1. Functionality requirements

The main functionality and modules of ReleasePlanner™ 2.0 is described in the following table

Table 16: Functionality requirements for ReleasePlanner™ 2.0

#	Functionality	Process	Description
1	Prioritization of features	Prioritize	Prioritizing features on a nine-point scale. Principle: Vary criteria for fixed feature, then vary features within a group, the vary feature groups.
2	Analysis of feature priorities	Analyze	Based on the priorities assigned by stakeholders to features:
3	Consensus analysis between stakeholders	Analyze	Display of sets of features being
4	Consensus analysis between groups of stakeholders	Analyze	Display of sets of features being
5	Dashboard	Dashboard	Presentation of highlight results and most recent actions.

#	Functionality	Process	Description
6	Optimized release plan generation	Optimize	Presentation of release plans in real calendars. Which sets of features are delivered when?
7	Comparison between two release plans	Analyze	For two selected plans, a comparison is made in terms of
8	Excitement analysis	Analyze	Stakeholder excitement analysis for strategic planning. Finding out expected excitements and disappointments of stakeholders with proposed plans.
9	Optimized staffing for the next release – Gant chart	Optimize	Generation of an optimized staffing plan, offered as a Gantt chart on a real-time calendar. Alternatively, for each developer, the sequence of tasks to be performed is presented.
10	Feature versus quality planning	Optimize	Generation of trade-off plans for balancing effort allocated to the development of new features versus fixing bugs and allocating effort towards (quality) stabilization of the product development.

#	Functionality	Process	Description
11	When-to-Release (W2RP) Optimization	Optimize	For a sequence of release dates, calculation (from re-optimization) of best release plans. Starting with the first plan, all new plans are described explicitly and/or in terms of the relative changes made.
12	Risk analysis	Analyze	Up-front scenario playing including varying selected parameters such as the release date, capacities, weights of criteria and stakeholder. Performed before the plan gets implemented, changes are displayed and analyzed.
13	Reporting	Control	Making selected (by user) results available and downloadable from an assigned url.
14	Interactive release plan generation	Optimize	Capability to generate release plans manually by assigning features to releases.

#	Functionality	Process	Description
16	Re-planning	Optimize	For an ongoing release, and a set of new feature having arrived, and an existing operational plan, re-planning results in a revised plan, supposed to be better than the previous one.
17	Project monitoring	Control	Displaying the status of implementation of both strategic and operational plans: List of features:
18	Estimation	Estimate	Collective effort estimation. As a result, for all features the estimated effort is determined.
19	Advanced feature dependencies	Define	Editing of all kinds of feature dependencies.
20	Notifications and discussion	Discuss	Notifications and all forms of discussion and stakeholder communication.
	Suggestion of Feature	Define	Allow stakeholders to suggest Features
21	Import and export of project information	Define	Import and export of project information (including results) for a set of most frequently used API's.
22	Support	Help	Tutorials, guidelines, Q&A's, online help desk.

#	Functionality	Process	Description
23	Management of projects		Sort, select, create and delete projects.
24	Stakeholder management	Define	Creation, editing and view of stakeholders and their related information.
25	Project creation and editing	Define	Creation, editing and view of a project and the related information.
26	JIRA Integration	API	Import and Export to JIRA - Sync

The process workflow and the interaction between the different modules and processes are illustrated in the below figure

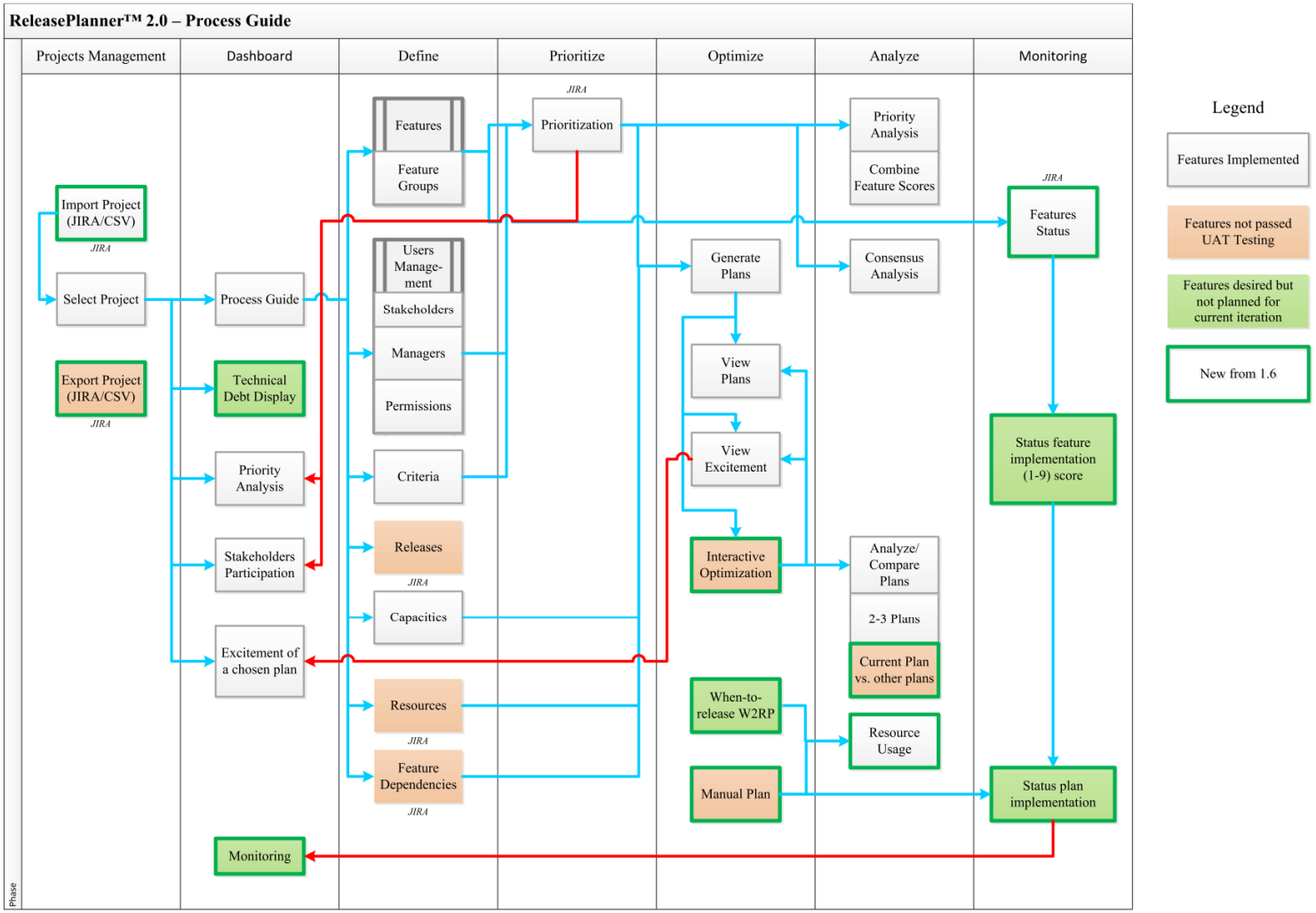

Figure 33: ReleasePlanner™ solution design

The features are scheduled to be delivered in the major release 2.0 on January 2015.

2. Testing

The development of ReleasePlanner™ 2.0 followed the test-driven-development (TDD) method, with Model-View-Controller (MVC) framework. Every new feature that was delivered had to pass the automated test suite before deployment was made. *Figure 34* outlined a test report from the development process.

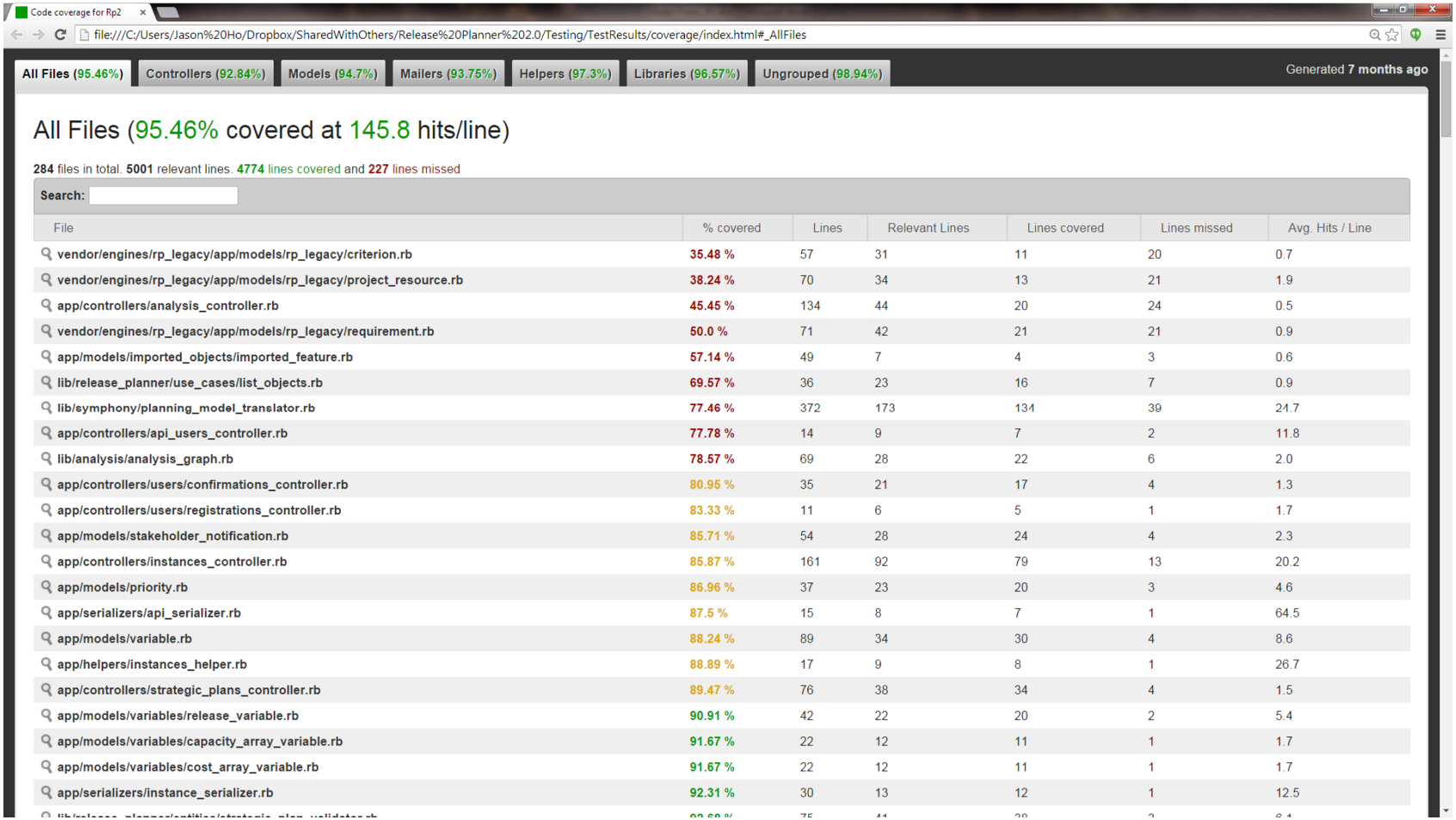

Figure 34: Test-driven-development test reports

Documentation, however, are tracked less formally through spreadsheets, JIRA repository issues comments, and mock-up screenshots. These documents, at the point of writing, were stored in a shared folder, yet not centralized. An example of how feature description is attached in issues management is presented below. There are no code or design documentation, as TDD is the practice of "source code is documentation".

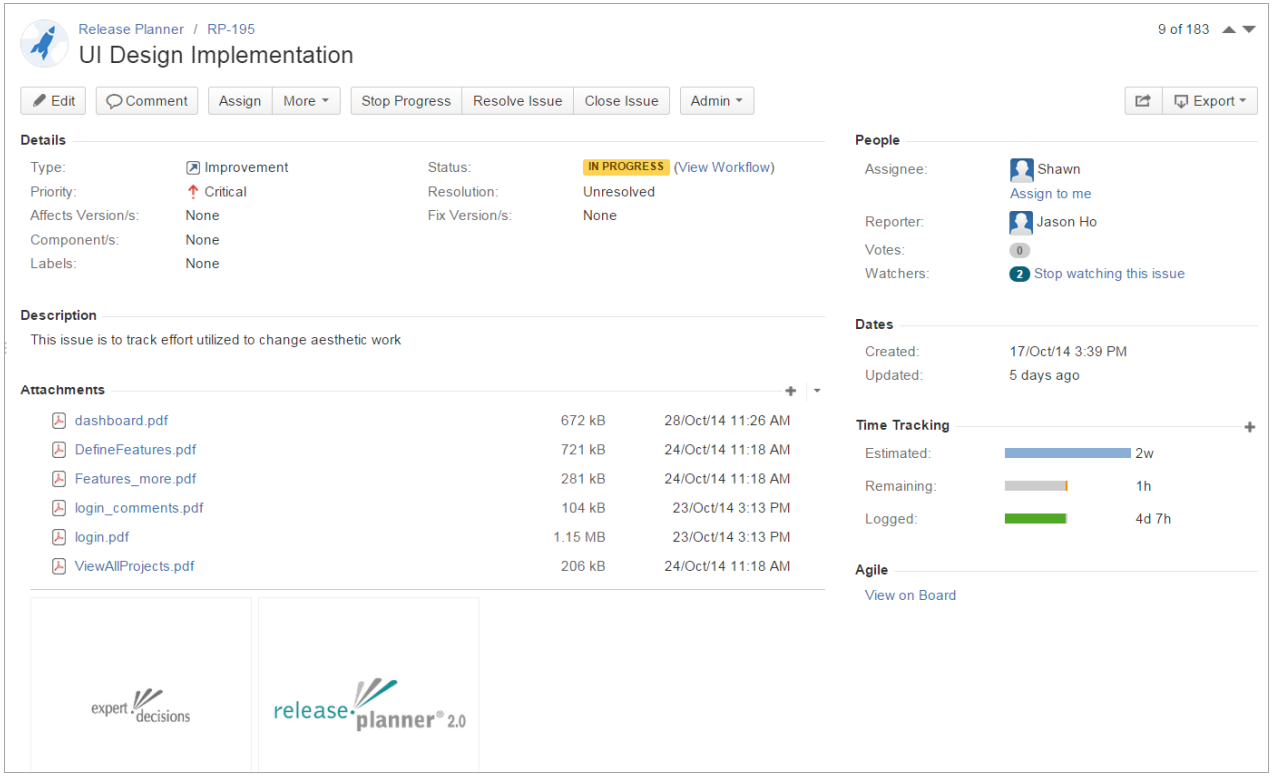

Figure 35: A feature being documented by JIRA

A. Technical debt evaluation

The evaluation and tracking of TD, after GQM were defined, were done via automated report from issue tracking JIRA, and manual evaluation of completeness. For each feature, we evaluated all metrics available. The final composite score is the combined fitness score of all features.

Functional Requirement	Complexity	Status	Design Document	Technical Specification	System Component(s)	Software Module(s)	Tested In	Complexity Score	%Doc Completion	%Code Completion	%Test Completion	%Fix Completion	Legacy?	Legacy size	Re-use Score	Total Doc Score	Total Code Score	Total Test Score	Total Fix Score
Prioritization of features	Very high	Completed	M100.2014	TD107.2014	Main frame	Prioritize		9.00	75%	100%	100%	100%	Y	50%	4.50	6.75	9	9	9
Analysis of feature priorities	Average	Testing	TDD Code	TD107.2014	Main frame	Analyze	UAT	5.00	100%	100%	90%	60%	Y	100%	5.00	5	5	4.5	3
Consensus analysis between	Average	Completed	TDD Code	TD107.2014	Main frame	Analyze	UAT	5.00	100%	100%	100%	100%	Y	100%	5.00	5	5	5	5
Dashboard	Very high	In	M101.2014	TD107.2014	Dashboard	Dashboard	UAT	9.00	100%	90%	90%	50%	N	0%	0.00	9	8.1	8.1	4.5
Optimized release plan generation	High	Testing	TDD Code	TD107.2014	Main frame	Optimize	UAT	7.00	100%	100%	100%	100%	Y	100%	7.00	7	7	7	7
Comparison between two release plans	High	In	M102.2014	TD107.2014	Main frame	Analyze		7.00	100%	100%	90%	100%	Y	25%	1.75	7	7	6.3	7
Excitement analysis	Average	Completed	TDD Code	TD107.2014	Main frame	Analyze	UAT	5.00	100%	100%	100%	100%	Y	100%	5.00	5	5	5	5
Optimized staffing for the next release	Very high			TD107.2014	Main frame	Optimize		9.00	0%	0%	0%	0%	Y	100%	0.00	0	0	0	0
Feature versus quality planning	High			TD107.2014	Main frame	Optimize		7.00	0%	0%	0%	0%	N	0%	0.00	0	0	0	0
Release time optimization	Very high			TD107.2014	Main frame	Optimize		9.00	0%	0%	0%	0%	N	0%	0.00	0	0	0	0
Risk analysis	Average			TD107.2014	Main frame	Analyze		5.00	0%	0%	0%	0%	N	0%	0.00	0	0	0	0
Reporting	Low			TD107.2014	Main frame	Control		3.00	0%	0%	0%	0%	Y	100%	0.00	0	0	0	0
Manual release plan generation	Low	Completed	TDD Code	TD107.2014	Main frame	Optimize	UAT	3.00	100%	100%	100%	100%	Y	100%	3.00	3	3	3	3
Re-planning	High			TD107.2014	Main frame	Optimize		7.00	0%	0%	0%	0%	N	0%	0.00	0	0	0	0
Project monitoring	Very high			TD107.2014	Main frame	Control		9.00	100%	0%	0%	0%	N	0%	0.00	9	0	0	0
Estimation	Average			TD107.2014	Main frame	Estimate		5.00	50%	0%	0%	0%	N	0%	0.00	2.5	0	0	0
Feature dependencies	High	Completed	M100.2014	TD107.2014	Main frame	Define	UAT	7.00	100%	100%	100%	100%	Y	100%	7.00	7	7	7	7
Notifications and discussion	Low	Completed	TDD Code	TD107.2014	Main frame	Discuss	UAT	3.00	100%	100%	100%	100%	Y	100%	3.00	3	3	3	3
Import and export of project information	Average	Testing	TDD Code	TD107.2014	Main frame	Define	UAT	5.00	75%	100%	100%	100%	Y	100%	5.00	3.75	5	5	5
Support	Very low			TD107.2014	Main frame	Dashboard		1.00	0%	0%	0%	0%	Y	100%	0.00	0	0	0	0
Management of projects	High	Completed	M100.2014	TD107.2014	Main frame	Define	UAT	7.00	100%	100%	100%	100%	Y	100%	7.00	7	7	7	7
Stakeholder management	Average	Completed	M100.2014	TD107.2014	Main frame	Define	UAT	5.00	100%	100%	100%	90%	Y	100%	5.00	5	5	5	4.5
Project creation and editing	Average	Completed	M100.2014	TD107.2014	Main frame	Define	UAT	5.00	100%	100%	100%	100%	Y	100%	5.00	5	5	5	5
Import Project from JIRA	High	Testing	M103.2014	TD99.2101	JIRA	JIRA	UAT	7.00	100%	100%	100%	80%	N	0%	0.00	7	7	7	5.6
Export Project to JIRA	High	In	M103.2014	TD99.2101	JIRA	JIRA		7.00	100%	50%	50%	50%	N	0%	0.00	7	3.5	3.5	3.5
Continuous Sync to JIRA	Very high	In	M103.2014	TD99.2101	JIRA	JIRA	UAT	9.00	75%	15%	50%	0%	N	0%	0.00	6.75	1.35	4.5	0

Figure 36: TD evaluation at feature-level

Dimension	Metrics	Weigh	Value	Composit	Membership score
Process rules compliance	Not available documentation in artefact	w_{d1-1}	0.2	0.02	0.4078
	Not available documentation in unit tests (TDD)	w_{d1-2}	0.5	0.05	0.1984
	The percentage of non-documented interfaces.	w_{d1-3}	0.3	0.03	0.0859
Quality testing	Number of defects found and not fixed at evaluation	w_{d2-1}	0.3	0.25	0.8687
	The percentage of code that is not tested, and not covered by automated tests	w_{d2-2}	0.6	0.2	0.4069
	The potential threat to security due to outdated, broken code	w_{d2-3}	0.1	0.05	0.1000
Maintainability	The interdependency between components	w_{d3-1}	0.4	0.08	0.8000
	The degree of platform or tool dependency	w_{d3-2}	0.4	0.08	0.8000
	The modular degree of code, the percentage of duplicate code	w_{d3-3}	0.2	0.04	0.5000
Complexity	Degree of legacy system and components available	w_{d4-1}	0.6	0.12	0.3953
	Current system completed code size	w_{d4-2}	0.4	0.08	0.4191
Factor b					**0.5532**

Figure 37: TDAF Calculation