A Requirements Measurement Program for Systems Engineering Projects: Metrics, Indicators, Models, and Tools for Internal Stakeholders

Ibtehal Noorwali, *The University of Western Ontario*

Supervisor: Nazim H. Madhavji, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Computer Science

Abstract

Software engineering (SE) measurement has shown to lead to improved quality and productivity in software and systems projects and, thus, has received significant attention in the literature, particularly for the design and development stages. In requirements engineering (RE), research and practice has recognized the importance of requirements measurement (RM) for tracking progress, identifying gaps in downstream deliverables related to requirements, managing requirements-related risks, reducing requirements errors and defects, and project management and decision making.

However, despite the recognized benefits of RM, research indicates that only 5% of the literature on SE measurement addresses requirements. This small percentage is reflected in the lack of well-defined and ready to use requirements metrics, approaches, tools, and frameworks that would enable the effective implementation and management of a RM program. Such a program would, in turn, provide the various internal stakeholders with various quantitative requirements-driven information (e.g., measures, indicators, and analytics, etc.) in order for them to better manage, control, and track their respective process activities. This shortage makes the process of RM, at best, complicated and, at worst, non-existent in most projects. The RM process is further complicated in large systems engineering projects due to large project sizes, numerous internal stakeholders, time pressure, large numbers of requirements, other software artifacts, to name a few.

This integrated-article thesis aims to address the aforementioned problem through the following main contributions that have been researched and validated within the context of a large systems engineering project in the rail-automation domain: (i) an empirically derived and validated structured requirements metric suite; (ii) an approach for deriving and organizing requirements metrics and related information; (iii) a requirements-centric, measurement-based health assessment framework; (iv) a meta-model for managing requirements -driven information for internal stakeholders; (v) a prototype requirements dashboard that builds upon and automates the concepts in i, ii, iii, and iv.

These contributions have implications for research on RM through extending the body of work on RM and promulgating further research. For practice, the results of this thesis are anticipated to facilitate the implementation and management of RM programs in real-world projects.

Keywords: Software and systems engineering, requirements engineering, measurement program, requirements metrics, empirical studies, internal stakeholders.

Summary for Lay Audience

Software requirements are descriptions of the capabilities, functions, services and constraints of a software. Requirements indicate how the software will work and interact with the user and what problems the software solves. The process of defining, documenting and maintaining the requirements of a software or system is called the *requirements engineering* (RE) process and is considered the first phase of the software engineering (SE) process. Requirements inform the subsequent software development phases and are used to design, develop (i.e., code), and test the software.

Large software projects are complex and difficult to manage. Thus, since the early days of SE, researchers and practitioners have attempted to measure software in order to better plan, control, organize, and improve software and the SE process. The SE literature is replete with metrics, measurement approaches and methods, metric thresholds, and measurement tools, to name a few. A combination of these measurement components form *measurement programs* that can be implemented in projects and organizations in order to enable the software measurement process.

However, much of the work focuses on measurement for the *design* and *development* phases of the SE process, in which the measured entity is *architecture* and *code* of a software. The work on measurement in the RE phase, in which software *requirements* are the measured entity, is limited despite evidence that shows that requirements measurement (RM) has benefits for the entire SE process such as reducing software defects, easier tracking of software development progress, better risk management, and improved project management and decision making.

This integrated-article thesis addresses this gap by proposing a RM program that consists of: i) a set of requirements metrics, ii) an approach for deriving and organizing requirements metrics, iii) a health assessment framework that integrates requirements measures with project data in order to define requirements-centric project health indicators, iv) a management aid for the RM process, and v) a requirements dashboard that automates the previous concepts.

The contributions of this thesis have implications for research and practice. For research, this thesis extends the limited body of knowledge on RM. In practice, organizations and projects can use the concepts in this thesis to implement a RM program.

Co-Authorship Statement

This integrated-article thesis contains several chapters that were co-authored by Ibtehal Noorwali and other authors. Ibtehal Noorwali is the principal author of all thesis chapters. Specifically, she formulated research ideas and questions, conducted the literature reviews, gathered and analyzed data, designed, built and validated solutions, and wrote and edited all thesis chapters.

Below, the contribution of the thesis supervisor, Professor Nazim H. Madhavji, to *all* thesis chapters (chapters 1–9) is first described. The contributions of each co-author are then described for each co-authored chapter (chapters 3–7).

All thesis chapters: Professor Madhavji's contribution to all thesis chapters involved fostering collaboration with researchers and practitioners, brainstorming research ideas, providing feedback on the design and implementation of solutions and studies, validating solutions and results, and discussing and reviewing each chapter.

Chapters 3 & 4: Dr. Remo Ferrari, from Siemens Mobility Inc., provided the opportunity to conduct the study on-site, was involved in identifying the research problem, and gave feedback on results throughout the study in Chapter 3. He provided all the necessary data (e.g., requirements, design documents, test cases, process and project documentation) for the studies in Chapters 3 and 4.

Chapter 5: Dr. Remo Ferrari and Matthew Dudycz, a former M.Sc. student at the University of Western Ontario, were involved in this study. As in Chapter 3, Dr. Ferrari was instrumental in the inception of the idea, providing data, and validating the results. Matthew Dudycz was involved in implementing the solution.

Chapter 6: Dr. Darlan Arruda and Dr. Remo Ferrari were co-authors of this chapter. Dr. Arruda, a former Ph.D. student at the University of Western Ontario helped in validating the meta-model, reviewing the chapter, and writing the threats to validity section. Dr. Ferrari took part in validating the meta-model.

Chapter 7: The principal author, Ibtehal Noorwali, managed the development of the system from its inception to deployment. Specifically, she specified the system's requirements and communicated them to the developers, explained underlying theoretical concepts, designed the system's interface, validated the interface design with users, aided in designing and architecting the system, developed algorithms and pseudo code for the back-end and front-end developers, tested the system, trained system users, and wrote and edited the chapter.

Elham Rahmani, Matthew Dudycz, Ross Johnston, Mikhail Jikharev, and Dr. Remo Ferrari also contributed to the system development. Particularly, Elham Rahmani, a M.Sc. student at the University of Western Ontario, helped in the system design stage. Matthew Dudycz set up the database and implemented the system's back-end. Ross Johnston and Mikhail Jikharev, undergraduate students at the University of Western Ontario, implemented the system's front-end. Dr. Ferrari helped in validating the system's requirements and evaluating the system.

Acknowledgements

{الْحَمْدُ لِلَّهِ الَّذِي هَدَانَا لِهَٰذَا وَمَا كُنَّا لِنَهْتَدِيَ لَوْلَا أَنْ هَدَانَا اللَّهُ}

No Ph.D. degree is a solo endeavour. I have been blessed with an incredible network of people who provided steady professional and personal support, guidance, and feedback and to whom I am eternally grateful.

First and foremost, I want to thank my supervisor, Professor Nazim Madhavji, who dared to take me on as a fresh, wide-eyed, and clueless master's student with no research experience to speak for. Many years, meetings, long discussions, debates, comments, and returned manuscripts later, I finally began learning the tricks of the trade and I am, I hope, a better researcher and writer for it. So thank you for your patience and guidance, for teaching me how good research is done, and for setting the academic standards so high.

I am also indebted to the many colleagues, mostly colleagues-turned-friends, who were instrumental in bringing this work to fruition.

Dr. Remo Ferrari, thank you for providing the opportunity to conduct this research, which without, I would probably still be floundering around, hunting for a research topic. Thank you for helping this academic navigate the foreign and largely unchartered territory—that is, industry. Thank you for the time, data, and feedback. But most importantly, thank you for the jokes, laughs and the constant stream of hockey talk—though unsolicited and mostly incomprehensible—which provided much needed respite from this serious and tedious business that is a Ph.D.

Dr. Darlan Arruda, I could not have asked for a better colleague and friend throughout this journey. You were there for it all; reviewing the papers, providing the feedback, attending the presentations, giving the pep talks, listening to the rants, witnessing the meltdowns, all while you had your own Ph.D. work to worry about. So thank you for being my solid rock through all the highs and lows.

I am grateful to all those who contributed to the content of the various chapters of this work—Elham Rahmani, Matthew Dudycz, Ross Johnston, Mikhail Jikharev, and many others—who went above and beyond what was required of them and who graciously put up with my frantic and ever-increasing demands to get things done in a deadline-induced frenzy. It has been nothing but a pleasure working with you and I hope I did not permanently damage the reputation of Ph.D. students.

My sincerest thanks to everyone who contributed to this work through reviewing, proofreading, providing feedback, filling out surveys, supplying data, establishing contacts, and helping, one way or another, in all those endless tasks that go into researching and writing a Ph.D. thesis. The list is far too long to include here, but I would like to particularly thank Philipp Hullmann, Lorna Stevenson, Dr. Irina Todoran Koitz, Eduard Groen, and Dr. Maya Daneva; your readiness to help was a life-saver and your contributions during this critical time will be long remembered.

I extend my thanks to the members of Western University and the Department of Computer Science for providing every possible accommodation, making this degree and project possible. Special thanks to the colleagues and friends at Siemens Mobility in New York for welcoming me into their midst with open arms and for motivating me to finish this degree by constantly reminding me that I need to find a more financially rewarding occupation.

This work would not have been possible without the generous funding by the Ministry of Higher Education of Saudi Arabia and Umm Al-Qura University, who funded the entirety of my graduate studies. I would also like to thank the Saudi Cultural Bureau in Canada for facilitating the funding process and ensuring that my studies and my time as a student are smooth-sailing. NSERC Canada also supported the last stretch of this work.

Finally, but certainly not because of least importance, I would like to thank my family and friends who supported me behind the scenes and cheered me on, knowingly or unknowingly, throughout this journey. Special thanks to my parents, Faten and Abdulsalam, my first teachers, for setting the bar of achievement incredibly high. I could not have done this without your prayers, your unwavering faith in me, and your regular shipments of coffee, sweets, and nuts from across the ocean.

My sisters, Essra and Asma, you are my heroes and inspirations. I only had to conjure up your simultaneous roles as mothers, graduate students, daughters, and sisters to give me the boost of motivation I needed. Witnessing your accomplishments, I really had no excuse—unburdened as I am—to not see this through. My brothers, Anas and Ahmad, I probably scared you away from pursuing a Ph.D. degree forever, but your constant, awestruck admiration of my academic struggles made me feel like Superwoman.

My heartfelt thanks to my friends—particularly Seereen, Heba, and Alyssa—who were always on stand-by, armed with love, food, words of comfort and motivation, and a ready ear for my tales of academic woes. You saw me at my best, but mostly at my absolute worst and, yet, somehow, you stuck around. To all the friends who were always ready for spur-of-the-moment road trips to wash away my academic blues, who indulged my stress-induced cravings, and who sent messages, postcards and gifts from all around the globe, thank you. The list is far too long to list but you know who you are and I am grateful for your presence.

It would be remiss of me not to acknowledge the substance that provided me with the dopamine, stamina, and illusion of confidence when I most needed it. So here's to the gallons of coffee I consumed throughout my Ph.D. time that helped me get the job done!

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Count what is countable. Measure what is measurable. And what is not measurable, make measurable. –Galileo Galilei.

The SE community has taken this advice to heart as measurement in systems and software engineering (SE) has received significant attention since the 1970s [Staron and Meding, 2018]. The significant effort dedicated to software measurement can be attributed to the many benefits accrued from using system and software metrics, including but not limited to: improved process productivity [Pfleeger, 1993], improved product quality [Fenton and Bieman, 2015], reduced cycle times and costs [Daskalantonakis, 1992; Gopal et al., 2002], and improved decision making [Johnson et al., 2005].

Measurement in SE is not confined to a simple process of defining and implementing a set of metrics; it encompasses a set of metrics, indicators, methods, tools and roles which are built to provide software development teams, project managers, product managers and quality managers with accurate and efficient measures, tools and instruments. This "socio-technical system where the technology interacts with stakeholders" in order to support measurement is termed a *measurement program* [Staron and Meding, 2016, 2018].

However, implementing measurement programs is not a simple task. The success of a measurement program for a specific process, project, or product is dependent on many factors such as the availability of a set of well defined metrics [Pfleeger, 1993; Gopal et al., 2002], automating the measurement process [Pfleeger, 1993; Johnson et al., 2005; Fenton and Bieman, 2015; Staron and Meding, 2018], and having well-defined procedures in place for managing the measurement process [Ebert and Dumke, 2007]. Thus, the efforts to facilitate the successful deployment and implementation of measurement programs have been significant and constant including defining [Chidamber and Kemerer, 1994] and validating [Kitchenham, 1995; Briand et al., 1995] software metrics, defining metric derivation procedures [Basili et al., 1994], understanding stakeholders' information needs [Buse and Zimmermann, 2012], building measurement tool support [Sillitti et al., 2003; Ohira et al., 2004; Johnson, 2007; Sharma and Kaulgud, 2012; López et al., 2018], and establishing metric thresholds and benchmarks [Jones, 2000; Ferreira et al., 2011a], among others.

In requirements engineering (RE), requirements measurement (RM) is a means to facilitate the requirements management process and can be defined as the process of collecting, analyzing, and reporting quantitative data relevant to the requirements through a set of metrics that enable the tracking and control of requirements and providing quantitative insight into the state of system development that would aid the different internal stakeholders in accomplishing their process-related tasks [Costello and Liu, 1995; Wiegers, 2006; IEEE, 2017].

RM has shown to have numerous benefits such as tracking development progress, identifying gaps in the downstream deliverables related to requirements [Kratschmer, 2013], managing requirements volatility and risks that may be introduced due to late changes to requirements [Costello and Liu, 1995; Kratschmer, 2013], reducing requirements errors and defects through quality control [Davis et al., 1993; Costello and Liu, 1995], and aiding in project management and decision making [IEEE, 2017]. Thus, without RM, projects risk undetected or unresolved requirements defects, lack of insight on progress, creeping project and product scope, unchecked requirements volatility, and ad-hoc decision making, all of which have negative impact on project cost, quality, time, and effort [Jones and Bonsignour, 2012; Ferreira et al., 2011b; Park et al., 2010; Damian and Chisan, 2006].

Despite the recognized benefits of RM by researchers and practitioners, we find that the body of work on RM is limited and that the majority of SE measurement work revolves around code-centric measurement [Kitchenham, 2010; Gómez et al., 2008]. A systematic literature review on the state of the art on software measurement reveals that only 5% of the literature addresses RM [Gómez et al., 2008] while design and development account for 42% and 27% of the measurement literature, respectively. A deeper analysis of the literature on RM reveals that there is a lack of: clear and well-defined requirements metrics, measurement approaches that deal with the specific challenges of RM, RM tool support, formal definition of RM entities, processes, and tasks, to name a few. According to Tahir et al. [Tahir et al., 2016], these factors hinder the successful implementation of measurement programs.

Our experience with industrial practitioners involved in large rail automation development projects provided us with a real-world manifestation of the above problem. Simply put, the RE team needs to provide, as part of the requirements management process, insightful and up-to-date requirements-driven information to downstream (e.g., design and testing) and side-stream processes (e.g., project management and quality management) that would help the internal stakeholders involved in these processes to manage, control, and track their activities. For example, designers need to know requirements-design traceability percentage in order to track their progress with regard to design. Similarly, project managers need to know the number of allocated requirements to a specific release that will allow them to manage time, costs, and budget accordingly. These examples underline the variety of stakeholder concerns that the requirements data needs to address. The problem lies in the lack of readily available metrics, analytical methods, and tools that would facilitate the requirements measurement process and provide the various stakeholders with insightful measurements and indicators that would inform their processes. Large project sizes, numerous stakeholders, large numbers of requirements and software artifacts, time pressure, unclear stakeholder concerns, and others, further exacerbates the problem.

While RM is not a completely new practice [Costello and Liu, 1995; Davis et al., 1993; Wiegers, 2006], the lack of an extensive body of work on requirements metrics, methods, and tools hinders the implementation of a requirements measurement program, and thus, risks losing the benefits of RM. This thesis addresses this problem, which we discuss in piecemeal in the following section.

1.1 Research Problem

As discussed above, the central problem is the lack of requirements metrics, methods, and tools that would facilitate the RM process. In this section, we detail the specific research problems that this thesis deals with based on our analysis of the literature and our experience with RM in practice.

P1. A lack of a set of well-defined and validated requirements metrics that can be used to address the different internal stakeholders' concerns. The problem with the requirements metrics in the literature are twofold: (i) they are vague and not well-defined (see below for example) and (ii) they mostly focus on requirements quality attributes such as complexity, size of an individual requirement, understandability and so forth [Génova et al., 2013; Antinyan and Staron, 2017].

M1: Requirements Completeness: Indicates completeness of all sections of a requirements specification, whether all allocated higher level requirements are addressed, and the degree of decomposition and the degree of decomposition of allocated higher-level requirements. [Costello and Liu, 1995]

M2: Size of requirements [Loconsole, 2001]

M3: Number of requirements traced or not traced [Kolde, 2004]

P2. A lack of a metric derivation method that addresses the specific challenges of RM. Despite our utilization of the GQM approach [Basili et al., 1994], there remained other challenges such as unorganized measures, incomplete metrics, and missing requirements meta-data items.

P3. A lack of understanding of how to incorporate requirements measures with other project data (e.g., deadlines, team size, and costs) to provide internal stakeholders, particularly managers, with a requirements-centric project health indicators.

P4. A lack of understanding of the web of interactions among all the entities (e.g., internal stakeholders, metrics, and processes) involved in the requirements measurement process. In other words, questions such as "Which stakeholders need what requirements-driven information?" and "What artifacts are the different measures derived from?" are left anunswered.

P5. As discussed above, one of the main inhibitors to software measurement is a lack of tool support. Current measurement tools focus on code-centric measures and requirements management tools have limited RM functionality. Thus, RM tool support is lacking in the literature and practice.

We note that the above problems are not by any means an exhaustive list of the problems faced in requirements measurement. However, due to the thesis scope and time limitations, we chose these problems that we believe would facilitate the implementation of requirements measurement programs within organizations.

1.2 Research Goal and Objectives

As motivated by the research problems discussed in Section 1.1, Figure 1.1 depicts the research goal of the thesis and the research objectives (RO) to achieve the stated goal. Each research objective addresses one of the research problems discussed in Section 1.1, respectively.

1.3 Contributions

The cumulative contribution of this thesis is a combination of empirical findings, techniques, models, and tools that, together, form a requirements measurement program for systems engineering projects. The thesis contributions are as follows:

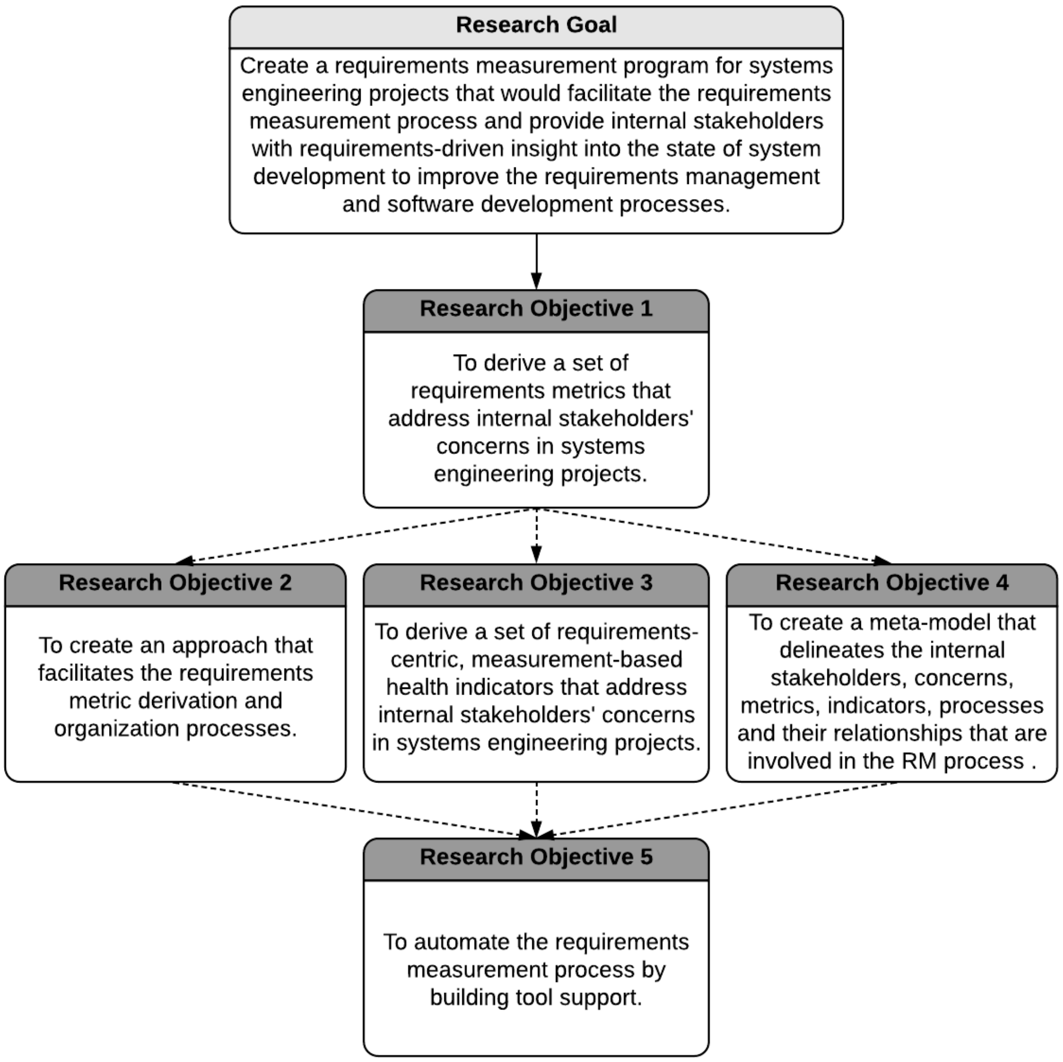

Research Goal
Create a requirements measurement program for systems engineering projects that would facilitate the requirements measurement process and provide internal stakeholders with requirements-driven insight into the state of system development to improve the requirements management and software development processes.
Research Objective 1
To derive a set of requirements metrics that address internal stakeholders' concerns in systems engineering projects.
Research Objective 2
To create an approach that facilitates the requirements metric derivation and organization processes.
Research Objective 3
To derive a set of requirements-centric, measurement-based health indicators that address internal stakeholders' concerns in systems engineering projects.
Research Objective 4
To create a meta-model that delineates the internal stakeholders, concerns, metrics, indicators, processes and their relationships that are involved in the RM process .
Research Objective 5
To automate the requirements measurement process by building tool support.

Figure 1.1: Thesis research goal and objectives.

C1. Empirical derivation of a structured requirements metrics suite from a large systems engineering project.

C2. Theoretical and empirical validation of the derived requirements metrics.

C3. Identification of the requirements attributes, levels, and meta-data that structure the derived requirements metrics.

C4. Creation of an approach for defining, analyzing, and organizing requirements metrics and related information.

C5. Creation of a health assessment framework that utilizes the derived requirements metrics in conjunction with project data to derive high-level, requirement-centric project health indicators.

C6. Operationalization of the health assessment framework based on real-world data from a systems engineering project.

C7. Identification of the entities (e.g., processes, internal stakeholders, metrics, indicators, and internal stakeholder concerns) and relationships (e.g., manages, is-used-in, and is-derived-from) involved in providing requirements-driven information to internal stakeholders.

C8. Construction and validation of a meta-model at three-levels of abstraction that combines the identified entities and relationships.

C9. Construction of a web-based requirements dashboard prototype that automates the metrics, health indicators, and meta-model concepts and that is to be deployed in a systems engineering project.

1.4 Thesis Roadmap and Chapter Overview

This thesis is in an integrated-article format that consists of five core chapters (Chapters 3-7). Each core chapter addresses one of the research objectives (RO) described in Section 1.2 and is a standalone article. Figure 1.2 depicts the the overall thesis roadmap. Particularly, it shows each core chapter's title, publication (if any), the research objective it addresses, the chapter's contributions as described in 1.3, and the relationships among the chapters. We provide an overview of each chapter below.

Chapters 1 and 2: Chapters 1 and 2 set the scene for the core chapters of the thesis. Chapter 1 describes the research problem (Section 1.1), associated research goal, and objectives (Section 1.2). In order to better understand the work in this thesis, it is necessary to describe the context in which the studies took place and for which the solutions in this thesis have been proposed. Thus, Chapter 2 describes the industrial context upon which the work in this thesis is based. Specifically, it includes the description of the company, studied projects, the RE process within those projects, the requirements management tool used, and the internal stakeholders.

Chapter 3: This chapter constitutes the foundation of this thesis as it describes the action research (AR) study [Santos and Travassos, 2011] that we conducted in the collaborating

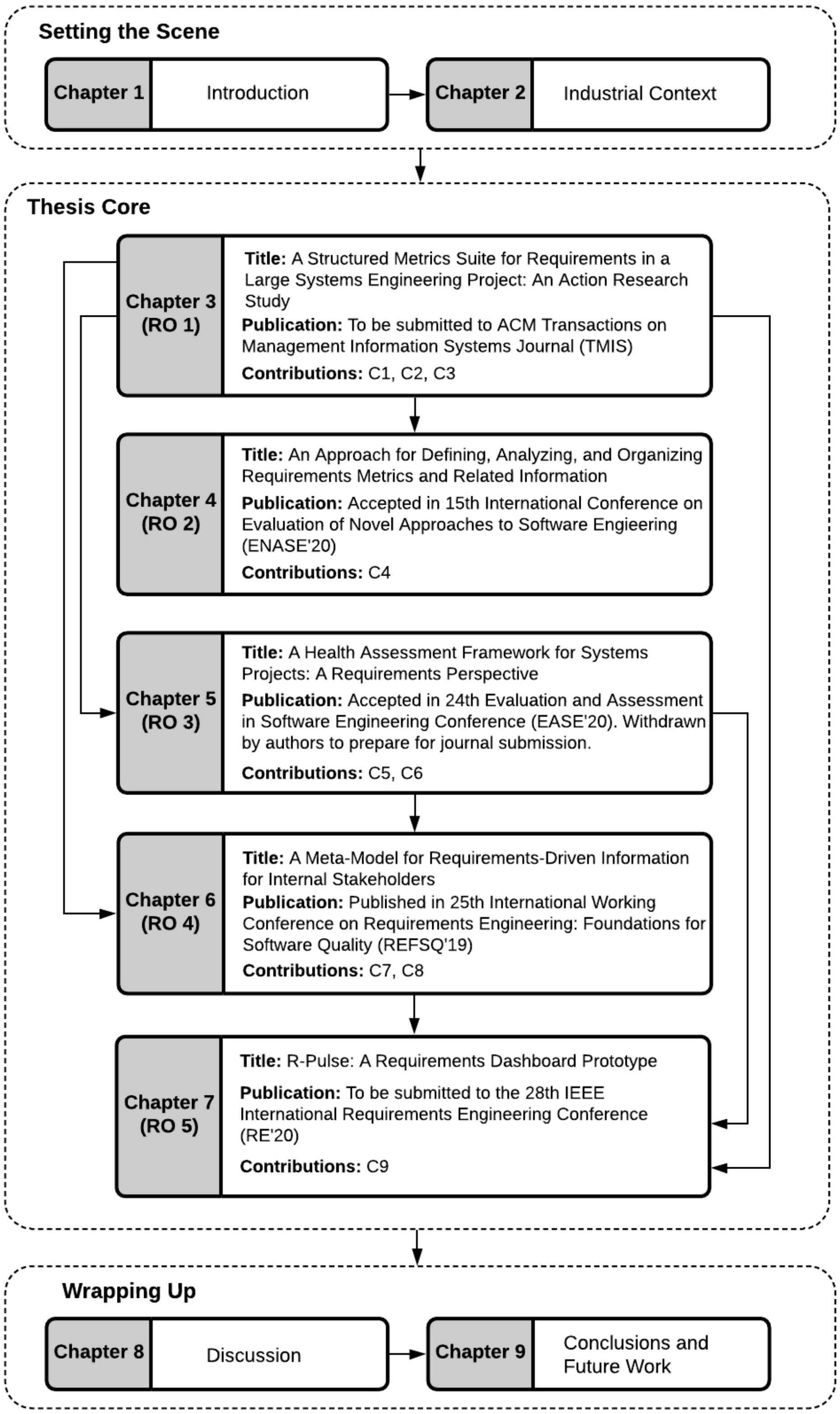

Setting the Scene
Chapter 1
Introduction
Chapter 2
Industrial Context
Thesis Core
Chapter 3 (RO 1)
Title: A Structured Metrics Suite for Requirements in a Large Systems Engineering Project: An Action Research Study
Publication: To be submitted to ACM Transactions on Management Information Systems Journal (TMIS)
Contributions: C1, C2, C3
Chapter 4 (RO 2)
Title: An Approach for Defining, Analyzing, and Organizing Requirements Metrics and Related Information
Publication: Accepted in 15th International Conference on Evaluation of Novel Approaches to Software Engieering (ENASE'20)
Contributions: C4
Chapter 5 (RO 3)
Title: A Health Assessment Framework for Systems Projects: A Requirements Perspective
Publication: Accepted in 24th Evaluation and Assessment in Software Engineering Conference (EASE'20). Withdrawn by authors to prepare for journal submission.
Contributions: C5, C6
Chapter 6 (RO 4)
Title: A Meta-Model for Requirements-Driven Information for Internal Stakeholders
Publication: Published in 25th International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ'19)
Contributions: C7, C8
Chapter 7 (RO 5)
Title: R-Pulse: A Requirements Dashboard Prototype
Publication: To be submitted to the 28th IEEE International Requirements Engineering Conference (RE'20)
Contributions: C9
Wrapping Up
Chapter 8
Discussion
Chapter 9
Conclusions and Future Work

Figure 1.2: Thesis roadmap.

organization. The goal of the AR study was to derive a requirements metric suite to: i) facilitate the requirements management process through enabling the tracking, monitoring, and management of requirements and requirements related-information and ii) provide internal stakeholders with requirements-related information that would address their concerns and aid them in their respective process activities. The results of this study includes a requirements metrics suite that consists of a set of 90 requirements metrics and 9 requirements meta-data items needed to apply the metrics. The metrics measure five requirements attributes (size, growth, volatility, status, and coverage) at four requirement metric levels (baseline, feature, release, and safety). The metrics are validated empirically through the AR study and theoretically using a software metrics validation framework.

Chapter 4: The large systems engineering context in which we conducted the AR study of Chapter 3 introduced unique challenges to requirements measurement: large sets of requirements across many sub-projects, requirements existing in different categories (e.g., hardware, interface, and software, etc.), varying requirements meta-data items (e.g., ID, requirement type, and priority, etc.), to name few. Consequently, the initial requirements measurement process was an ad-hoc one that lead to incomplete metrics, unorganized metrics and measurement reports, and incomplete and missing meta-data items that are essential to applying the the metrics. Thus, in Chapter 4 we present a 7-step approach that combines the use of GQM [Basili et al., 1994] and the identification and analysis of four main RE measurement elements: attributes, levels, metrics, and meta-data items that aid in the derivation, analysis, and organization of requirements metrics. We illustrate the use of our approach by applying it to further projects from the rail automation systems domain. We show how the approach led to a more comprehensive set of requirements metrics, improved organization and reporting of metrics, and improved consistency and completeness of requirements meta-data across projects.

Chapter 5: Upon defining the metrics described in Chapter 3, there was a need within the studied project to assess project health based on a combination of the derived metrics and project data (e.g., deadlines, schedule, and costs) and provide high-level health indicators for each project. However, there is a lack of systematized and measurement-based approaches that explicitly factor requirements into their analyses. Thus, we address this gap in Chapter 5 by proposing a requirements-centric project health assessment framework that measures and analyzes critical requirements attributes, in conjunction with project data, and identifies their health indicators, which are visualized using a RED-AMBER-GREEN (RAG) indicators system. The implemented framework is applied to three real-life systems projects. The evaluation results demonstrate the feasibility of the framework and a level of agreement with human assessors who have evaluated the projects' health.

Chapter 6: Providing the requirements-driven information (e.g., requirements volatility measures, requirements-design coverage information, and requirements growth rates, etc.) identified in Chapters 3 and 5 falls within the realm of the requirements management process. The requirements engineer must derive and present the appropriate requirements information to the right internal stakeholders in the project. This process was made complex due to project-related factors such as numerous types of internal stakeholders, varying stakeholder concerns with regard to requirements, project sizes, a plethora of software artifacts, and many affected processes. There is little guidance in practice as to how these factors come into play together in providing the described information to the internal stakeholders. Thus, based on analyzed data from the AR study, Chapter 6 presents a meta-model that consists of the main entities and relationships involved in providing requirements-driven information to internal stakeholders within the context of a large systems project. The meta-model consists of five main entities and nine relationships that are further decomposed into three abstraction levels. We validated the meta-model in three phases by researchers and practitioners.

Chapter 7: The findings and solutions from Chapters 3, 4, 5, and 6 would not be fully utilized in industry without some level of automation. To this end, Chapter 7 presents R-Pulse, a web-based requirements dashboard prototype that automates the concepts from the previous chapters. R-Pulse uses exports of the requirements documents and other software artifacts (e.g., design, test, and defects) to calculate the requirements metrics and health indicators. Internal stakeholders are able to view the project's health indicators and drill down to the requirements measures. The dashboard provides various visualization and navigation options so as to address each internal stakeholder's need with regard to the requirements-driven information.

Chapter 8 and 9: Chapters 8 and 9 wrap up the thesis. Particularly, Chapter 8 situates the measurement program in terms of practice and literature, discusses a summary of the validation of the findings and solutions presented in the core chapters, and evaluates the robustness of the overall measurement program. Finally Chapter 9 concludes the thesis and describes future work.

References

[Antinyan and Staron, 2017] Antinyan, V. and Staron, M. (2017). Rendex: A method for automated reviews of textual requirements. *Journal of Systems and Software*, 131:63–77.

[Basili et al., 1994] Basili, V., Caldiera, G., and Rombach, H. (1994). Goal question metric approach. *Encyclopedia of Software Engineering*, 1:98–102.

[Briand et al., 1995] Briand, L., El Emam, K., and Morasca, S. (1995). Theoretical and empirical validation of software product measures. Technical report, International Software Engineering Network.

[Buse and Zimmermann, 2012] Buse, R. P. L. and Zimmermann, T. (2012). Information needs for software development analytics. In *34th International Conference on Software Engineering (ICSE 2012)*, pages 987–996, Zurich, Switzerland. IEEE.

[Chidamber and Kemerer, 1994] Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

[Costello and Liu, 1995] Costello, R. J. and Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems Software*, 29:39–63.

[Damian and Chisan, 2006] Damian, D. and Chisan, J. (2006). An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Transactions on Software Engineering*, 32(7):433–453.

[Daskalantonakis, 1992] Daskalantonakis, M. K. (1992). A Practical view of software measurement and implementation experiences within Motorola. *IEEE Transactions on Software Engineering*, 18(11):998–1010.

[Davis et al., 1993] Davis, A., Overmeyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., and Theofanos, M. (1993). Identifying and measuring quality in a software requirements specification. In *Proceedings of the 1st International Software Metrics Symposium*, pages 141–152, Baltimore, MD, USA. IEEE.

[Ebert and Dumke, 2007] Ebert, C. and Dumke, R. (2007). *Software measurement: establish, extract, evaluate, execute*. Springer.

[Fenton and Bieman, 2015] Fenton, N. E. and Bieman, J. (2015). *Software metrics: a rigorous and practical approach*. CRC Press, 3rd edition.

[Ferreira et al., 2011a] Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F., and Almeida, H. C. (2011a). Identifying thresholds for object-oriented software metrics. *The Journal of Systems & Software*, 85:244–257.

[Ferreira et al., 2011b] Ferreira, S., Shunk, D., Collofello, J., Mackulak, G., and Dueck, A. (2011b). Reducing the risk of requirements volatility: findings from an empirical survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 23:375–393.

[Génova et al., 2013] Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., and Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41.

[Gómez et al., 2008] Gómez, O., Oktaba, H., Piattini, M., and García, F. (2008). A systematic review measurement in software engineering: State-of-the-art in measures. In *Proceedings of the 3rd International Conference on Software and Data Technologies (ICSOFT'08)*, pages 165–176.

[Gopal et al., 2002] Gopal, A., Krishnan, M. S., Mukhopadhyay, T., and Goldenson, D. R. (2002). Measurement programs in software development: Determinants of success. *IEEE Transactions on Software Engineering*, 28(9):863–875.

[IEEE, 2017] IEEE (2017). ISO/IEC/IEEE 15939:2017(E) - Systems and software engineering - Measurement Process. Technical report, ISO/IEC/IEEE.

[Johnson, 2007] Johnson, P. M. (2007). Requirement and design tradeoffs in Hackystat: An inprocess software engineering measurement and analysis system. In *1st International Symposium on Empirical Software Engineering and Measurement*, pages 81–90, Madrid, Spain. IEEE.

[Johnson et al., 2005] Johnson, P. M., Kou, H., Paulding, M., Zhang, Q., Kagawa, A., and Yamashita, T. (2005). Improving software development management through software project telemetry. *IEEE Software*, 22(4):76–85.

[Jones, 2000] Jones, C. (2000). *Software assessments, benchmarks, and best practices*. Addison-Wesley.

[Jones and Bonsignour, 2012] Jones, C. and Bonsignour, O. (2012). *The economics of software quality*. Addison-Wesley.

[Kitchenham, 1995] Kitchenham, B. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944.

[Kitchenham, 2010] Kitchenham, B. (2010). What's up with software metrics? - A preliminary mapping study. *Journal of Systems and Software*, 83(1):37–51.

[Kolde, 2004] Kolde, B. C. (2004). Basic metrics for requirements management. Technical Report March, Borland.

[Kratschmer, 2013] Kratschmer, T. (2013). Requirements and metrics. *Requirements Management Blog - IBM*. www.ibm.com/developerworks/community/blogs/requirementsmanagement/entry/requirements-metrics?lang=en.

[Loconsole, 2001] Loconsole, A. (2001). Measuring the requirements management key process area. In *Proceedings of the 12th European Software Control and Metrics Conference (ESCOM'01)*, pages 67–76, London, UK.

[López et al., 2018] López, L., Martínez-Fernández, S., Gómez, C., Choraś, M., Kozik, R., Guzmán, L., Vollmer, A. M., Franch, X., and Jedlitschka, A. (2018). Q-rapids tool prototype: Supporting decision-makers in managing quality in rapid software development. In *Lecture Notes in Business Information Processing*, volume 317, pages 200–208.

[Ohira et al., 2004] Ohira, M., Yokomori, R., and Sakai, M. (2004). Empirical project monitor: A tool for mining multiple project data. In *Proceedings of the International Workshop on Mining Software Repositories (MSR'04)*, pages 42–46, Edinburgh, UK.

[Park et al., 2010] Park, S., Maurer, F., Eberlein, A., and Fung, T.-S. (2010). Requirements attributes to predict requirements related defects. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, pages 42–56, Toronto, ON. ACM.

[Pfleeger, 1993] Pfleeger, S. L. (1993). Lessons learned in building a corporate metrics program. *IEEE Software*, 10(3):67–74.

[Santos and Travassos, 2011] Santos, P. S. M. d. and Travassos, G. H. (2011). Action research can swing the balance in experimental software engineering. *Advances in Computers*, 83:205–276.

[Sharma and Kaulgud, 2012] Sharma, V. S. and Kaulgud, V. (2012). PIVoT: Project insights and visualization toolkit. In Zurich, S., editor, *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM'12)*, pages 63–69. IEEE.

[Sillitti et al., 2003] Sillitti, A., Janes, A., Succi, G., and Vernazza, T. (2003). Collecting, integrating and analyzing software metrics and personal software process data. In *Proceedings of the 29th EUROMICRO Conference*, pages 336–342, Belek-Antalya, Turkey. IEEE Computer Society.

[Staron and Meding, 2016] Staron, M. and Meding, W. (2016). MeSRAM - A method for assessing robustness of measurement programs in large software development organizations and its industrial evaluation. *Journal of Systems and Software*, 113:76–100.

[Staron and Meding, 2018] Staron, M. and Meding, W. (2018). *Software development measurement programs: Development, management and evolution*. Springer.

[Tahir et al., 2016] Tahir, T., Rasool, G., and Gencel, C. (2016). A systematic literature review on software measurement programs. *Information and Software Technology*, 73:101–121.

[Wiegers, 2006] Wiegers, K. E. (2006). *More about software requirements: Thorny issues and practical advice*. Microsoft Press, Washington.

Chapter 2

The Industrial Context

To provide context to the research problems discussed in Section 1.1 and the findings and solutions to be presented in the following chapters, we describe the industrial context in which this work was conducted. Particularly, we describe the company and the projects in terms of size and domain, the projects' requirements engineering process, the requirements management tool used within the project, and the types of internal stakeholders.

2.1 Company and Projects

The collaborating company is a multinational company in the transportation industry specializing in railways, control systems and digital services with over 30,000 employees. This study was conducted in a large-scale rail automation systems project in an American-based branch. The overall project (i.e., program) consisted of multiple sub-projects, three of which we were directly involved with. The program aims to provide products, solutions and services to the North American freight and commuter market. Each sub-project consists of a product for a specific client and has its own set of requirements, architecture design, test cases, and engineering team. Table 2.1 shows the duration for each project and a breakdown of their artifacts.[1]

Table 2.1: Descriptive statistics of projects.

Project	Project Duration	No. Req. Baselines	No. Reqs.	No. Safety Reqs.	No. Design Baselines	No. Design Objects	No. Test Cases
P1	73 months	54	1790	N/A	23	472	2111
P2	36 months	30	2285	N/A	4	380	N/A
P3	45 months	51	2389	923	28	827	2045

[1] For improved readability and easier navigation of the thesis, this table will also appear in other chapters of the thesis that refer to it.

2.2 Requirements Engineering Process

The requirements engineering (RE) process is adapted from the ISO/IEC/IEEE 29148-2011 International Standard for Systems and Software Engineering – Life cycle processes – Requirements engineering [IEEE, 2011] and an internal requirements engineering standard.

The RE process is embedded in other project processes. It has direct interfaces to *upstream* processes (e.g., contract/client management and change management), *downstream* processes (e.g., design, verification and validation , subcontractor management, installation, site management, manufacturing, and development), and *sidestream* processes such as training, project management, RAMSS (Reliability, Availability Maintainability, and Safety Security), development, and quality.

The purpose of the RE process is twofold:

1. To develop and maintain a shared understanding among the project team of the system to be implemented.

2. To establish and maintain traces from requirements to other project artifacts (i.e., design, development, and testing).

The RE process consists of the following activities: requirements elicitation, analysis, validation and management. Figure 2.1 depicts the RE process in the project.

We see in Figure 2.1 that *requirements metrics and indicators* (depicted in red) are an output of the requirements management activity. Given the focus of this thesis, we only describe the requirements management sub-process to understand how requirements measurement factors into the process.

The purpose of the requirements management subprocess is to record and maintain the evolving requirements and associated context and historical information from the requirements engineering activities and to establish procedures for defining, controlling, and publishing the baseline requirements for all levels of the system-of-interest. Requirements management consists of the following activities:

- **Setting project and module structure:** Involves setting the directory, project, and module structure in the requirements management tool according to project and product hierarchy.

- **Managing requirements meta-data:** Requirements meta-data are descriptive properties associated with a requirement (e.g., ID, created by, created on, release number, safety, type, and status), which can be used to make the decision-making process

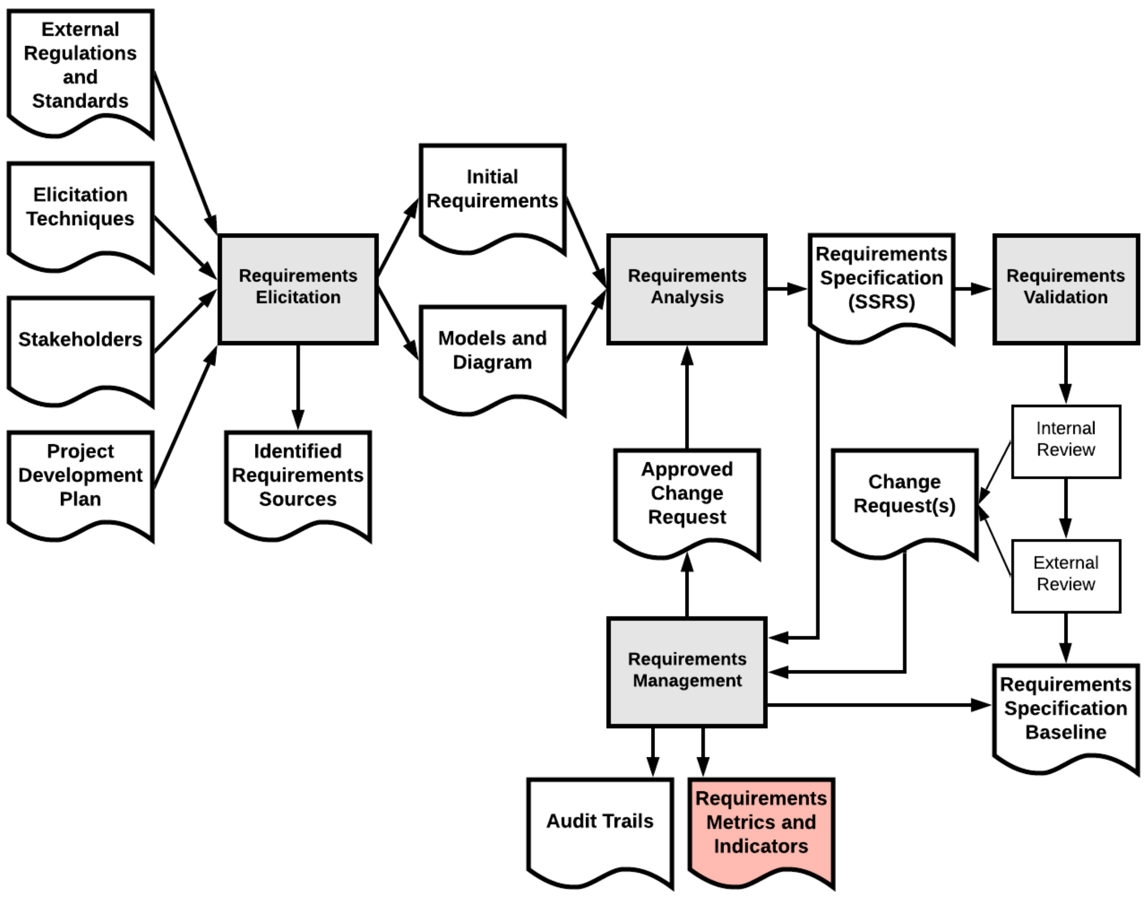


Figure 2.1: The organization's requirements engineering process.
Depicted in this figure are the activities/subprocesses of the organization's RE process (rectangles) and the input and outputs to the the activities. The RE process is adapted from the ISO/IEC/IEEE 29148-2011 Requirements Engineering standard [IEEE, 2011] and an internal requirements engineering standard.

more objective. The meta-data items and their values must bet managed in the requirements management tool. The values assigned to each meta-data item help to organize, analyze, and prioritize the requirements in each project.

- **Managing requirements workflow states:** This is accomplished by managing the requirement meta-data item that indicates where the requirements state at a particular time is against the expectation of what "complete" means for the requirements process. State transitions are set manually.

- **Tracing:** Ensures that requirements are traceable throughout all phases, documents, and components of the product lifecycle. A requirements tracing model is used for conducting impact analyses, tracking project development progress (metrics), proving that all requirements have been satisfied by the design, and that test artifacts have been developed and executed to satisfy all requirements.

- **Baselining requirements:** A requirements baseline is a set of requirements that stakeholders have agreed to, often defining the contents of a specific planned release or development iteration. In the requirements management tool, a baseline is a static read-only version of a module, corresponding to a snapshot of the module at a specific time. Frozen versions (e.g., release versions) of requirements or objects in general are created as baselines.

- **Maintaining audit trails:** The histories of each module and requirement are stored in the requirements management tool. Creation date, module/requirement author, modification dates, modification author, baseline dates, and deletion dates are stored and can be accessed through the requirements management tool.

- **Requirements measurement:** Involves defining metrics, data collection methods, data collection, data analysis, and data reporting in order to produce requirements measures and indicators to track requirements progress, state of traceability, and requirements change, to name a few.

- **Requirements change management:** Changes and exceptions to a baseline will be handled by a Change Control Board (CCB). The CCB will evaluate the impact and cost of the change or exception, assign resources to investigate and resolve, and establish the effectiveness of the solution. The solutions will be implemented in accordance with the normal engineering processes (e.g., document control and requirements engineering).

2.3 Requirements Management Tool

The Rational Dynamic Object Oriented Requirements System (DOORS) [IBM, 1991] is used to store and manage the requirements and design objects. DOORS is a requirements management tool that provides a set of features to capture, trace, analyze, and manage changes to information. Rational DOORS facilitates internal stakeholders' participation and contribution to the requirements management process through managing changes to requirements and allowing collaboration through requirements *discussions*. It also provides functionalities for linking requirements to design items, test plans, test cases, and other requirements for easy and powerful traceability.

2.4 Internal Stakeholders

Each project consisted of the following roles: requirements engineers, requirements manager, project managers, architects, safety managers, quality managers, verification and validation managers, developers, testers, hardware manager, tool support engineers. Overseeing all the projects within the program is the *program director*. Table 2.2 lists some of the projects' internal stakeholders and describes their responsibilities with regard to the requirements.

References

[IBM, 1991] IBM (1991). IBM engineering requirements management DOORS family. See: https://www.ibm.com/ca-en/marketplace/requirements-management.

[IEEE, 2011] IEEE (2011). ISO/IEC/IEEE 29148 - Systems and software engineering — Life cycle processes — Requirements engineering. Technical report, IEEE.

Table 2.2: Projects' internal stakeholders and responsibilities.

Internal Stakeholder	**Responsibilities**
Systems Engineer (requirements engineer)	– Develops and maintains the Requirements Management Plan – Elicits requirements with customers and other stakeholders – Develops and manages requirements – Manages and maintains the requirements baselines, including the requirements database – Defines and reports requirements metrics for project RE progress – Defines and executes the requirements change management process
Program Director	– Making decisions about requirements scope and contractual requirements for the program – Provides input on feature prioritization
Project Manager	– Develops and monitors the schedule for the requirements elicitation phase – Establishes allocation of requirements to development iterations – Supports coordination of requirements elicitation and validation activities among project stakeholders – Ensures that contractual requirements are met
Architect	– Ensures system design meets requirements – Supports elicitation and analysis of requirements (especially non-functional requirements and other architecturally significant requirements) – Assesses requirements for feasibility – Reviews and approves requirements – Maintains traceability between requirements and design
Safety and Security Manager	– Assigns/reviews safety attributes of requirements – Provides input on reliability, maintainability, availability and security requirements – Reviews and approves requirements
Quality Manager	– Reviews and approves the requirements documents – Ensures that established requirements process is followed
Verification & Validation Manager	– Reviews and approves requirements – Develops V&V methods and artifacts against requirements – Maintains traceability between requirements/design and V&V artifacts
Software Development Manager	– Implements the software requirements – Reviews and approves requirements
Hardware Manager	– Constructs products based on hardware requirements – Supports definition of installation requirements – Establishes manufacturing process
Tools Support Engineer	– Customizes and supports the integrated tool set – Implements requirements tooling requirements – Develops integration script and supports configuration management efforts – Manages tool licenses – Coordinates tool training

Chapter 3

A Structured Metrics Suite for Requirements in a Large Systems Engineering Project: An Action Research Study

3.1 Introduction

Requirements measurement (RM) is part of the requirements management process and we define it as *the process of collecting, analyzing, and reporting quantitative data relevant to the requirements through a set of metrics* [Costello and Liu, 1995; Wiegers, 2006; IEEE, 2011]. The metrics (e.g., # of requirements/baseline, % of requirements added per feature per baseline, and # of requirements covered by design) enable the tracking and control of requirements and provide quantitative insight into the state of system development that would aid the different internal stakeholders in accomplishing their process-related tasks. The benefits of using requirements metrics includes, but not limited to: (1) tracking progress, (2) identifying gaps in the downstream deliverables related to requirements [Kratschmer, 2013], (3) managing risks that may be introduced due to requirements changes [Costello and Liu, 1995; Kratschmer, 2013], (4) reducing requirements errors and defects through quality control [Davis et al., 1993; Costello and Liu, 1995], and (5) aiding in project management and decision making [IEEE, 2011].

Despite the numerous benefits of requirements measurement, requirements metrics have received little attention in the literature. In a systematic literature review of software engineering metrics, the authors found that only 5% of the papers discussed were in the analysis (i.e., requirements) phase and the majority were in the design (42%), development (27%), maintenance (14%), and testing (12%) phases [Gómez et al., 2008]. Based on a survey of the literature (Section 3.2) and our experience with requirements metrics in industry (Section 3.3), we identified *four* major problems with requirements metrics.

First, there is a lack of *well-defined* and *validated* requirements metrics in the literature that can be readily applied in practice and that could be used by the various internal stakeholders within a project to enable the tracking and control of requirements. In particular, requirements metrics usually consist of vague descriptions of how to carry out the measurements without any description of the formulas needed to calculate the measures [Costello and Liu, 1995; Berenbach and Borotto, 2006; Wiegers, 2006].

Second, literature that discusses requirements metrics [Costello and Liu, 1995; Loconsole, 2001; Wiegers, 2006] does not discuss the data needed to apply the metrics. As a simple example, *the number of requirements that have been approve for implementation* [Wiegers, 2006] would need at least two requirements meta-data items: a unique requirement ID and a *status*. However, proposing metrics without specifying the meta-data needed for applying the metrics will inhibit the adoption of proposed metrics [Costello and Liu, 1995; Kratschmer, 2013].

Third, there is a lack of understanding as to the measurable *requirement attributes* that the metrics in the literature are measuring. This lack of understanding regarding the measurable requirements attributes can lead to inaccurate metric definitions (i.e., defining metrics that are not measuring what they are purporting to measure) and a difficulty in validating the metrics because most validation frameworks rely on having an unambiguous understanding of the attributes being measured [Kitchenham, 1995; Briand et al., 1995, 1996]. Moreover, due to the dearth of research on measurable requirements attributes, we do not know what measurable requirement attributes could yield useful metrics that would aid the requirements management process in a large systems projects and provide useful requirements information to the various internal stakeholders. The literature has largely focused on requirement quality (e.g., individual requirement size [Génova et al., 2013], complexity [Antinyan and Staron, 2017], and consistency [Byun et al., 2014], etc.) and volatility metrics [Loconsole and Börstler, 2005; Kulk and Verhoef, 2008; Ferreira et al., 2011].

Finally, requirements in a large-scale systems project are large in number, and, thus, exist at different levels or categories [IEEE, 2011]. For example, a set of requirements may be organized as a whole in requirement baselines (i.e., requirement document versions), which are further organized within the baseline according to features and releases. Research shows that different requirement levels may trigger different requirements-information needs for internal stakeholders, which, in turn, demand different set of metrics [Gross and Doerr, 2012a; Hess et al., 2017]. However, the literature is silent in this regard; we do not know what requirements levels could yield useful requirements metrics.

Thus, the goal of this paper is to address the above issues through identifying an empirically derived and validated requirements measurement suite that consists of requirements attributes, metric-levels, and associated metrics and meta-data to: i) facilitate the requirements management process through enabling the tracking, monitoring, and management of requirements and requirements related-information and ii) provide internal stakeholders with requirements-related information that would address their concerns and aid them in their respective process activities.

The contributions of this article are as follows: (1) We bring to light five measurable requirement attributes (i.e., size, growth, status, volatility, and coverage) (Section 3.4.1) that are important to measure in a systems engineering project and that have received varying degrees of attention in the literature and practice, (2) we identify four requirement metric levels (i.e., baseline, feature, release, and safety) (Section 3.4.2) that exist in large systems projects, (3) we identify the meta-data items needed to apply the defined metrics (Section 3.4.3), (4) we present an empirically derived and validated set of 90 requirements metrics at all attribute-level combinations (e.g., requirements size metrics at the baseline level) (Section 3.4.4), and (5) we validate the metrics empirically through an action research study and theoretically using an established measurement validation framework [Kitchenham, 1995] (Section 3.5.1).

The research was conducted as an action research (AR) study in a large systems project in the rail automation domain (Section 3.3). The project consists of three sub-projects each of which consists of 1500+ requirements and has its own team. We defined the requirements metrics in collaboration with the internal stakeholders at the organization and empirically validated them through incorporating the metrics into requirements management process in a semi-automated way. We show how the measures at the different levels can be used to address different internal stakeholder concerns through example usage scenarios from the projects.

In practice, our results show that the identified requirement metrics for the different attributes and levels can be used within requirements management and in downstream and side-stream processes to manage requirements, gain insight into system development through requirements-driven information and to aid decision-making within the project (Section 3.5.1). Theoretically, the results enhance our understanding of requirements as measurable software entities and contribute to the limited body of literature on requirements metrics.

Moreover, the results are anticipated to have further implications for practice and research (Section 3.6) through improving requirements engineering documentation practice, increasing requirements measurement breadth, forming a foundation for metric tools and applications (e.g., requirements dashboards), and promulgating further research on measurable requirements attributes, levels, and metrics.

3.2 Related Work

In this section, we discuss the related work surrounding the four elements of our research: measurable requirements attributes (e.g., volatility) (Section 3.2.1), requirements levels (e.g., feature) (Section 3.2.2), requirements metrics (e.g., % of requirements added per feature per baseline) (Section 3.2.3), and finally requirements meta-data (e.g., requirements type) (Section 3.2.4) for requirements measurement. We then analyze and highlight the research gap in Section 3.2.5.

3.2.1 Measurable Requirements Attributes

An attribute is a property of an entity [Fenton and Biemen, 2015]. For instance, a person is an entity and height is one of the person's attributes. Thus, a requirement attribute is a property that a requirement or set of requirements (i.e., entity) possesses. Examples of such requirement attributes include size [Wiegers, 2006; Loconsole and Borstler, 2007], growth [Jones, 1996b], volatility [Zowghi and Nurmuliani, 2002], completeness [Costello and Liu, 1995], and consistency [Byun et al., 2014], etc. Attribute-based measurement has been widely adopted in software engineering [Davis et al., 1993; Morasca and Briand, 1997; Fenton and Biemen, 2015] and acquiring a clear and unambiguous understanding of the attributes being measured is essential for accurate measurements [Chidamber and Kemerer, 1994; Briand et al., 1996]. Thus, identifying the attributes for the real-world entity (i.e., requirements) that we wish to measure is the first step in the measurement process [Morasca and Briand, 1997; Fenton and Biemen, 2015].

The software engineering literature is replete with studies on software "code-centric" attributes such as coupling, cohesion, size, and complexity [Briand et al., 1996] and their associated metrics. In RE, there exists a plethora of studies on requirement *quality* attributes. Davis et al. [Davis et al., 1993] make one of the first attempts to define requirements quality attributes that can be measured. They identify 24 qualities such as *unambiguous, complete, correct,* and *concise* (see Table 3.1 for complete list) and propose measures for 18 of the attributes. Similarly and more recently, Genova et al. [Génova et al., 2013] discuss a set of requirements quality attributes that consist of validability, verifiability, modifiability, completeness, traceability, among others (see Table 3.1 for complete list) and the requirements indicators that can be used to measure the quality attributes (e.g., size, punctuation, connective terms, and degree of nesting, etc.). While the previous studies focused on a group of requirements quality attributes, other studies focus on one quality attribute as is the case with [Byun et al., 2014] who study requirements *consistency* and propose metrics for measuring it. Other studies have taken a code-centric approach to requirements metrics by adopting studied code quality attributes (e.g., coupling, cohesion, and complexity, etc.) and using said attributes as a basis for proposing requirements metrics [Antinyan and Staron, 2017]. Other requirements attributes that have been touched upon, but not explored in depth, include defect density [Costello and Liu, 1995], status [Berenbach and Borotto, 2006; Wiegers, 2006; Ebert and Dumke, 2007], and traceability [IEEE, 1989; Costello and Liu, 1995; Goti, 1998; Nassar and Scandariato, 2017].

One of the most widely studied requirements attributes is requirements volatility. Work on requirements volatility is multi-faceted and includes: metrics for requirements volatility [Loconsole and Börstler, 2005]; the causes [McGee and Greer, 2012] and effects of requirements volatility on other elements such as project performance [Zowghi and Nurmuliani, 2002; Ferreira et al., 2009] and defects [Malaiya and Denton, 1999]; benchmarking requirements volatility rates [Jones, 2000; Kulk and Verhoef, 2008]; managing requirements volatility and associated risks [Thakurta and Ahlemann, 2010; Ferreira et al., 2011]; and predictors of requirements volatility [Loconsole and Borstler, 2007].

Table 3.1: Summary of literature survey on requirements metrics.

Study	**Measurable Requirement Attribute**	**Metric Definition**	**Metric Levels**	**Meta-Data**	**Validation**	**Domain**
S1 [Costello and Liu, 1995]	Volatility, coverage, completeness, defect density, fault density, interface consistency, problem report/action item/issue, integrated process	General descriptions of how to measure the attributes	Generic levels: product, process, progress, resource, system and software	N/A	N/A	Large software-intensive systems
S2 [Wiegers, 2006]	Product size, quality, status, requests for changes (volatility), effort	General descriptions of how to measure the attributes	N/A	Discussed the *status* meta-data	N/A	N/A

S3 [Antinyan and Staron, 2017]	Requirement quality (complexity, coupling, size, length, cohesion)	5 metrics	N/A	Requirement text	Empirical (Expert opinion, regression analysis, correlation analyses)	Large software development organization
S4 [Génova et al., 2013]	Requirement quality (validability, verifiability, modifiability, completeness, consistency, understandability, unambiguity, coverage, abstraction, precision, atomicity)	General description of how some of the attributes can be measured	N/A	N/A	Empirical (tool)	Tool used in different domains
S5 [Byun et al., 2014]	Requirements consistency (quality)	5 metrics	N/A	N/A	Sample scenario	N/A
S6 [Berenbach and Borotto, 2006]	Progress, status, completeness, quality	11 metrics	N/A	Brief mention of meta-data needed for 2 metrics	N/A	2 large systems projects, 1 large software project
S7 [Kolde, 2004]	Not explicitly identified but includes: Volatility, defects rate, growth, status, coverage	General descriptions of 11 metrics	N/A	N/A	N/A	N/A
S8 [Kulk and Verhoef, 2008]	Growth	4 metrics and mathematical model	N/A	Function points	Empirical (case study)	Banc-assurance sector
S9 [Loconsole, 2001]	Not explicitly identified but includes volatility, size, defect rate, quality	51 metrics	N/A	N/A	N/A	N/A

S10 [Loconsole, 2003]	Requirements specification size, size of changes per requirement, status of requirements specifications, progress of change requests in life cycle, change classification, rationale for changes, duration, resource	9 metrics	N/A	Theoretical	N/A	N/A
S11 [Loconsole and Börstler, 2005]	UCM size (quality), volatility	5 volatility metrics, 4 UCM size metrics	N/A	Description of one meta-data item for one metric	Empirical (case study)	Small project – embedded software
S12 [Loconsole and Borstler, 2007]	Requirement size, volatility	Definition for requirements volatility only. General description for other metrics	N/A	One meta-data briefly discussed	Empirical	Small project – embedded software
S13 [Lam et al., 1999]	Dependency, change density, volatility, error rate, fix cost, acceptance rate	General descriptions of 7 metrics for the attributes	N/A	N/A	N/A	N/A
S14 [Goti, 1998]	Size, goodness, traceability, change	Descriptions of how to measure the attributes	N/A	Generic descriptions of use cases, events-in-flow, scenarios	N/A	N/A
S15 [Chemuturi, 2013]	Requirements stability, quality (defect injection rate, delivered defect density)	15 metrics	N/A	N/A	N/A	N/A

S16 [Davis et al., 1993]	24 SRS quality attributes (unambiguous, complete, correct, understandable, verifiable, internally & externally consistent, achievable, concise, design independent, traceable, modifiable, electronically stored, executable, annotated by importance, stability, and version, not redundant, right level of detail, precise, reusable, traced, organized, cross-referenced)	18 metrics	N/A	N/A	N/A	N/A
S17 [IEEE, 1989]	Traceability/Coverage, Compliance	2 metrics	N/A	Brief descriptions of *primitives*	N/A	N/A
S18 [Peterson and Wohlin, 2010]	Not explicitly identified but includes: coverage, size	4 metrics	N/A	N/A	Static validation	N/A
S19 [Nassar and Scandariato, 2017]	Traceability/Coverage	2 metrics	N/A	N/A	Empirical	Automotive industry
S20 [Ebert and Dumke, 2007]	Status, progress, quality	General descriptions of 7 ways to measure the attributes	N/A	Brief discussion of ID, description, effort, value	N/A	N/A
This study	Size, growth, volatility, status, coverage	90 metrics	Baseline, feature, release, safety	9 meta-data items	Empirical and theoretical	Large systems project in the rail automation domain

3.2.2 Requirements Metric Levels

The categorization of entities for measurement purposes is a common practice in SE. One of the most common software metric categorizations consists of product, process, resources [Ebert and Dumke, 2007; Fenton and Biemen, 2015], and progress [Costello and Liu, 1995] metrics. Others have categorized software engineering management metrics into project management level and general management level metrics [Offen and Jeffery, 1997]. Similarly, [Kerzner, 2017] identifies business and project levels for software metrics. Costello and Liu [Costello and Liu, 1995] intersect the product, process resources, and progress metric levels with system-level and software-level metrics. Such categorizations are different than identifying an entity's measurable attributes. For example, the *size* attribute on the software-level requires different metrics than the *size* attribute at the system-level. While the former may use LOC to measure the software size, the latter may use LOC in addition to hardware components to measure system size. However, the study of requirements levels for the purpose of measurement, to our knowledge, is nonexistent. Particularly, an explicit identification of the requirement metric levels that would address the requirements-related stakeholder concerns is nonexistent (see Table 3.1 for details) in the RE literature.

3.2.3 Requirements Metrics

In this subsection, we discuss the related work on requirements metrics and focus on two issues: the *definition* and *validation* of the metrics that have been proposed to measure the requirements attributes identified in our study (i.e., size, growth, volatility, status, and coverage). The IEEE Standard 1061 defines a metric as a *function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possess a given attribute* [IEEE, 1992]. Thus, a well-defined metric entails a well formulated function whose output is single numerical value. A poorly-defined metric (i.e., vague descriptions of metrics) inhibits the adoption of proposed metrics [Costello and Liu, 1995]. Ambiguous metric definitions may also lead to measuring attributes or entities other than what is intended. For example, a volatility metric that is defined as the amount of change that occurs in a requirement set is vague and may be open to several interpretations. Questions such as *what constitutes a change?* and *what constitutes a requirements set?* will significantly alter the resulting numerical value that is purported to measure requirements volatility. Meanwhile, metric validation consists of theoretical validation (i.e., demonstrating that the metric is really measuring the attribute it is purporting to measure) and empirical validation (i.e., demonstrating a metric's usefulness in a given context) [Briand et al., 1995]. Metrics that have not been validated theoretically and empirically are meaningless [Costello and Liu, 1995; Briand et al., 1995; Antinyan et al., 2016; Kitchenham, 2010].

Table 3.1 lists 20 studies from the RE literature that propose requirements metrics in one way or another. ***Requirements size*** metrics are metrics that attempt to measure the overall size of the requirements set and has been referred to as product size [Wiegers, 2006]. Five (S2, S9, S10, S12, S14) of the 20 studies have proposed requirements size metrics. S2 [Wiegers, 2006] stresses the importance of measuring product size to know the number of requirements in a product, but no metric definition is given. Loconsole et al. propose several requirements size metrics. In S9 [Loconsole, 2001], there is one undefined size metric that is referred to as *size of requirements* and another metric defined as *the total number of requirements,* which is not explicitly associated with *size.* The proposed metrics are not validated. In S10 [Loconsole, 2003], Loconsole and Borstler define the requirements specification size as *the total number of requirements,* with no further details on how to count the number of requirements. They also attempt to validate the metric theoretically and empirically, but the empirical validation is inconclusive. In S12 [Loconsole and Borstler, 2007], Loconsole and Borstler propose the *number of words per file, number of lines per file, number of uses cases per file,* and *number of actors per use case* as requirements size metrics. They attempt to validate the metric empirically by conducting an analysis to test whether the size metrics are good predictors of requirements volatility. S14 [Goti, 1998] proposes to measure *project size* by counting number of features and number of use cases. However, no metric definition or validation is provided in the paper.

Requirements growth or creep metrics measure the increase or decrease in the requirements overall size over time [Kulk and Verhoef, 2008]. S9 [Kulk and Verhoef, 2008] uses requirements *volatility* to refer to requirements creep and scrap (i.e., growth). However, their definition of volatility adheres to our definition of requirements growth and not volatility. Thus, S9 defines 4 requirements growth metrics and provides empirical validation for the metrics through a case study. In S7 [Kolde, 2004], no requirements growth metric definition is given, but it simply states that requirements growth over time metric *"can help the project manager determine whether adequate progress is being made gathering and specifying the requirements. As the project progresses, unusual growth can be an indicator of scope creep. It may also be an indicator that there are opportunities to improve the way in which requirements are elicited and documented."*

Requirements volatility metrics measure the change to requirements and have been given different definitions. In S1, Costello and Liu [Costello and Liu, 1995] propose to measure requirements volatility by counting additions, deletions, and modifications over a duration of time and classified by reason for change. No definition or validation of the proposed metrics are given. S2 [Wiegers, 2006] defines a requirements volatility metric as *Number of added requirements + number of deleted requirements + number of modified requirements / initial number of requirements.* No metric validation is provided. In S7 [Kolde, 2004], Kolde proposes to measure the amount of change in a project through the frequency of change in the total requirements set, rate of introduction of new requirements, and number of requirements changes to a requirements baseline. No metric definition or validation is provided in the paper. In the series of studies by Loconsole on requirements metrics, volatility metrics are addressed with varying degrees of detail. In S9 [Loconsole, 2001], volatility is not explicitly mentioned but several metric proposals for *requirements change* are given such as: the number of changes per requirement, the size of change per requirement, and number of changes to requirements per unit of time. No further details or metric validation is provided. In S10 [Loconsole, 2003], theoretical validation of 'size of change per requirement' is attempted. In S11 [Loconsole and Börstler, 2005], requirements volatility is measured by total number of changes to a use case model (UCM), number of minor changes, number of moderate changes, number of major changes, and number of revisions. A case study is conducted to validate the measures empirically. In S12 [Loconsole and Borstler, 2007], the volatility metric is defined as the *the amount of changes to a requirements document over time and measure it as the sum of the change densities of a requirements document.* Empirical validation is performed through an experiment. In S13 [Lam et al., 1999], *the number of requirements that have been added, modified or removed within a given reporting period as a proportion of the total number of requirements* is suggested to measure volatility. In S14 [Goti, 1998], Goti proposes to track requirements change through *how many features were modified.* No further definition or validation is provided in S13 and S14. Finally, S15 [Chemuturi, 2013] proposes the following formula to calculate *requirements stability*: (number of change requests / number of requirements) * 100.

Requirements status metrics aid in in tracking the status of requirements in a project over time. S2 [Wiegers, 2006], S6 [Berenbach and Borotto, 2006], S7 [Kolde, 2004], S10 [Loconsole, 2003], and S20 [Ebert and Dumke, 2007] propose to have a *status* attribute for each requirement and then a count of each status to indicate the overall status of requirements. However, the studies do not include metric definition nor validation. Moreover, there is no agreement as to which statuses should be assigned (e.g., analyzed, agreed, tested, or reviewed).

Finally, ***requirements coverage***, sometimes referred to as traceability, metrics attempt to measure how much of the requirements have been covered by design, implementation, and tests. Five studies (S4, S7, S17, S18, S19) out of the 20 studies in Table 3.1 discuss requirements coverage metrics whether implicitly or explicitly. S4 [Génova et al., 2013] suggests counting the number of dependencies towards other requirements or other artifacts to measure coverage. No metric definition or validation is provided. S7 [Kolde, 2004] simply discusses the importance of tracking *the number of requirements traced or not.* S17 [IEEE, 1989] defines the coverage metric as *(the number of requirements met by the architecture/ number of original requirements)*100.* S18 [Peterson and Wohlin, 2010] simply states that metrics at the requirements level should include the number of requirements in design, implementation and test. Finally, S19 [Nassar and Scandariato, 2017] defines two metrics for the number of traced requirements per component and traced components per requirements, which are validated empirically through an experiment that uses the coverage metrics to predict software defects.

3.2.4 Requirements Meta-Data

Good requirements engineering practice dictates that requirements must have a set of defining *attributes* or *meta-data* [IEEE, 2011]. Although there is no hard and fast rule as to what meta-data must be maintained for requirements as different organizations have different needs, it is common practice that each requirement should have a unique ID, has a clear description (i.e., text), type, status, priority, and rationale [Kotonya and Sommerville, 1998; Wiegers, 2006; IEEE, 2011]. However, a set of requirements metrics would require a specific set of requirement meta-data in order to be able to derive the metrics and calculate the measurements. Out of the 20 sources we found on requirements metrics, S2, S3, S6, S11, S12, S14, and S17 contain brief implicit descriptions of the meta-data that can be used in the proposed metrics (e.g., requirements text, requirements status, and function points, etc.). S20 [Ebert and Dumke, 2007] included an explicit, brief discussion of the meta-data required for the requirements metrics they propose (i.e., ID, description, effort, and value).

3.2.5 Analysis and Research Gap

In the previous subsections, we examined the literature on requirements metrics from several angles: measurable requirements attributes, requirement metric levels, requirement metric definition and validation, and requirements meta-data. Table 3.1 summarizes the surveyed sources and includes the measurable requirement attribute(s), how metric definition(s), metric levels, and meta-data are addressed, existence and type of validation of the metrics, and the context from which the metrics were derived. We discuss the research gap based on Table 3.1 and our discussions in the previous subsections.

First, the measurable requirements attributes that have received the most attention are requirements quality and volatility, with volatility having been studied from many angles (see Section 3.2.1). However, other requirements attributes, and particularly those that need to be measured within the context of large systems engineering projects, are understudied. This article addresses this gap by bringing to light the requirements attributes that are prevalent in large-sale systems engineering projects.

Second, our discussion in Section 3.2.2 and Table 3.1 show that work on requirements metric levels is nearly nonexistent. This article, to our knowledge, is the first of its kind to examine requirements metric levels.

Third, Section 3.2.3 demonstrates how the majority of metrics for RE lack the rigor that metrics in SE in general have been subject to. In other words, the bulk of RE metrics are vague descriptions and, in the case that a proper metric definition is provided, lacks the necessary validation for it to be useful in practice. In a recent and extensive mapping study of SE metric papers, Kitchenham [Kitchenham, 2010] shows that the most cited papers are analysis and validation metric papers. However, the majority of the evaluation metric papers evaluate object-oriented metrics, while RE metrics are not discussed at all. This article addresses this gap not by proposing novel metrics per se, but by bringing a higher level of rigor to the metrics so as to ensure their usefulness and usability in practice.

Fourth, as discussed in Section 3.2.4, the gathering of meta-data for measurement is essential to the measurement process. However, meta-data is seldom discussed in RE metric papers (see *Requirement Meta-Data* column in Table 3.1). Our work identifies the necessary requirements meta-data items for the proposed metrics. Finally, the last column in Table 3.1 shows that the context from which the majority of metrics were derived is unknown or mostly software intensive projects (expect for S1 and S6), which casts doubt upon the metrics' applicability to a large systems project. Meanwhile, the results in this paper are empirically derived from and validated within a large systems project.

3.3 Research Methodology

The research was initiated in a large-scale rail automation project in a multi-national company in the United States as an action research (AR) study. Action research (AR) is a process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action planning, intervention/action taking, evaluation, and reflection/ learning [Susman and Evered, 1978], where researchers identify problems through close involvement with industrial projects, and create and evaluate solutions in an almost indivisible research activity. Thus, AR provides practical value to the collaborating organization while simultaneously contributing to the acquisition of new and robust theoretical knowledge [Easterbrook et al., 2008]. Particularly, action research has shown to be effective in deriving and validating SE metrics [Antinyan et al., 2016; Staron, 2019].

Our AR study followed the canonical approach [Davison et al., 2012]. To report the action research study, we followed the guidelines by [Davison et al., 2012] and template provided by [Santos and Travassos, 2011]. Figure 3.1 presents the AR cycle performed (adapted from [Susman and Evered, 1978]). We discuss the project (context, participants, and data) and the AR procedure in the following subsections.

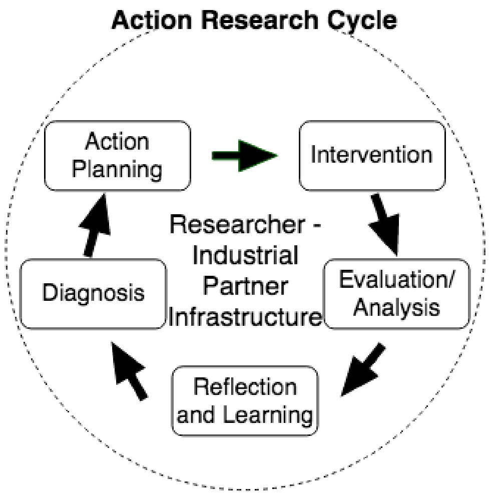


Figure 3.1: The action research approach adopted in the study.

3.3.1 Descriptive Statistics: Project Context, Data, Participants

Project Context

The AR study began in February 2017 with the primary researcher being onsite full-time for ten months and worked with the primary industrial partner and secondary industrial participants (internal project stakeholders) in consultation with a senior researcher. The overall project (i.e., program) consisted of three sub-projects, each sub-project consisted of a product that had its own set of requirements, architecture design, test cases, and engineering team. As mentioned earlier, the sub-projects are part of a large-scale rail automation project in a multi-national company in the United States. The project adopted a waterfall software development approach. The requirements engineering process generally consisted of requirements elicitation, analysis, specification, and validation and management stages and adhered to an internal requirements engineering standard.

Project Data

Table 3.2 shows a breakdown of the software artifacts which consists of the number of requirements baselines (i.e., document versions), requirements, design baselines, design objects, and test cases per product that the first author worked with. The Rational Dynamic Object Oriented Requirements System (DOORS) was used to store and manage the requirements and design objects. IBM Engineering Test Management (formerly RQM) was used to store and manage test cases. All the requirements and design baselines and test cases for all products were exported to spreadsheets.

Table 3.2: Descriptive statistics of projects.

Project	Project Duration	No. Req. Baselines	No. Reqs.	No. Safety Reqs.	No. Design Baselines	No. Design Objects	No. Test Cases
P1	73 months	54	1790	N/A	23	472	2111
P2	36 months	30	2285	N/A	4	380	N/A
P3	45 months	51	2389	923	28	827	2045

Participants

The primary industrial partner is the systems manager who overlooks the requirements engineering and management processes for all the projects under study. The other internal project stakeholders included: R&D managers, test mangers, developers, architects, testers, project managers, program managers, safety managers, quality mangers, financial managers, and project operations managers, all with whom the researcher worked with during the AR study.

3.3.2 Action Research Procedure

An AR study typically consists of five phases and each phase consists of a set of activities. Table 3.3 summarizes our AR study and includes the theoretical AR activities that must take place in each phase (second column), the corresponding real-world activities that took place (third column), the participants involved (fourth column) in each phase, the methods and tools used in each AR phase (fifth column), and the output of each phase (sixth column). We describe each phase in detail in the following subsections. Given the nature of AR, the questions are not predefined before the initiation of the study. Rather, the research questions emerge during the action planning phase as a result of the problem diagnosis phase and in the reflection and learning phase as a result of identifying empirical and theoretical findings [Santos and Travassos, 2011].

Table 3.3: Overview of action research procedure.

AR Phase	**AR Activities (theory)**	**Real-World Activities (conduct)**	**Participants**	**Tools and Methods**	**Outputs**
Phase 1: Diagnosis	Identify problem	Established collaboration with organization, worked onsite as part of the RE team	Researcher, primary industrial participant, senior researcher	Unstructured interviews	Problem statement and scope
Phase 2: Action Planning (incl. Research Questions)	Explore potential solutions, select solution, conduct literature survey, define research questions	Research questions defined in this study	Researcher, primary industrial participant, senior researcher	Unstructured interviews, document analysis, literature survey	Proposed solution, research questions, related work
Phase 3: Intervention/Action Taking	Collect and analyze data, design solution, insert solution into process, gather feedback on solution, update/ enhance solution based on feedback	Identify meta-data, define metrics, use metrics in project, update metrics based on stakeholder feedback	Researcher, primary industrial participant, secondary industrial participants (internal project stakeholders)	Document analysis, GQM, unstructured interviews, focus groups, observation	Requirements metrics, meta-data
Phase 4: Evaluation/ Analysis	Analyze intervention effects	Validate metrics empirically	Researcher, primary industrial participant, secondary industrial participants (internal project stakeholders)	Observation, informal discussions, interview	Requirements metrics, meta-data
Phase 5: Reflection and Learning	Identify theoretical and empirical findings and lessons learned, define further research questions	Validate metrics theoretically, identify attributes, identify levels, identify meta-data	Researcher, primary industrial participant, secondary industrial participants (internal project stakeholders), senior researcher	Content analysis, Measurement validation framework	Requirements, metrics, attributes, levels, meta-data

Phase 1: Diagnosis

In the diagnosis phase the researcher and primary industrial participant (i.e., systems manager) held unstructured interviews to understand the project context, requirements engineering process, software development process, and the involved internal project stakeholders. Unstructured interviews are those in which there is no specific set of predetermined questions, but with certain topics in mind that need to be covered; they flow like an everyday conversation and tend to be more informal and open-ended [Thorpe and Holt, 2008]. Through the unstructured interviews and briefing sessions, the researcher and primary industrial participant identified the general problem being faced in the requirements management process: *difficulty in tracking, monitoring, and managing requirement-related information such as the number of requirements in a project, number of requirements per software release, requirements growth and volatility rates, etc. and difficulty to access this information by internal project stakeholders.*

Phase 2: Action Planning

In this phase the researcher and primary industrial participant, through further unstructured interviews, selected a solution to the problem identified in the diagnosis phase that is to be designed, implemented and incorporated into the requirements management process: *a set of requirement-centric metrics that will be made available to the internal project stakeholders.* The primary participant provided a brief description of what requirements-related information need to be monitored: number of requirements over requirements baselines, number of modified requirements, number of safety critical requirements, and number of requirements per feature. Equipped with the understanding of available data and the needs of the systems manager, the researcher conducted a literature survey to explore the requirements metrics in the literature that can be used in the requirements management process. However, as described in Section 3.2, the literature provided little guidance on requirements metrics.

Thus, based on the participant's information concerns, document analysis, and literature survey, the primary researcher defined the goal of the AR study: *to identify an empirically derived and validated set of requirements metrics that can be used by internal stakeholders in a systems engineering project to gain insight into their respective process activities, which, in turn, would aid the requirements management and other systems development processes.* The following research question was initially identified for the AR study:

RQ1: What requirements metrics are appropriate for use by various project internal stakeholders in a large systems engineering project, such that it provides them with insights satisfying their specific concerns?

The researcher and primary industrial participant held briefings to discuss the requirements meta-data available in the requirements repositories that can be possibly used to apply the metrics. The researcher then conducted an initial document analysis of requirement specification documents, architectural design documents, and test documentation to understand the available meta-data and the feasibility of deriving the above information. Thus, the following research question was posed:

RQ2: What requirements meta-data in a large systems engineering project are needed to enable the application of the requirements metrics identified in RQ1?

As discussed in Section 3.1, requirements metrics lacked theoretical validity as opposed to other SE metrics [Kitchenham, 2010]. Thus, in order to demonstrate the theoretical validity of the metrics, we began investigating measurement validation frameworks [Kitchenham, 1995; Schneidewind, 1992; Briand et al., 1995; Fenton and Biemen, 2015], all of which required identifying the entities that are being measured (i.e., requirements in our case) and their attributes (e.g., size, volatility, complexity, etc.). In consequence, we defined the following research question:

RQ3: What measurable requirements attributes are the metrics identified in RQ1 measuring so as to facilitate the validation of the metrics?

Finally, there was a need for further structuring of the measures in order to: 1) provide the internal stakeholders with the right requirements measures (i.e., which internal stakeholders needed which requirements measures) and 2) structure the measure reports in a uniform and readable format. Thus, it was necessary to organize the metrics into logical categories or groups, which led to the fourth research question:

RQ4: At what metric levels do requirements the metrics derived in RQ1 exist in a large systems engineering project?

Figure 3.2 depicts the hierarchy of the research questions and the relationships among them.

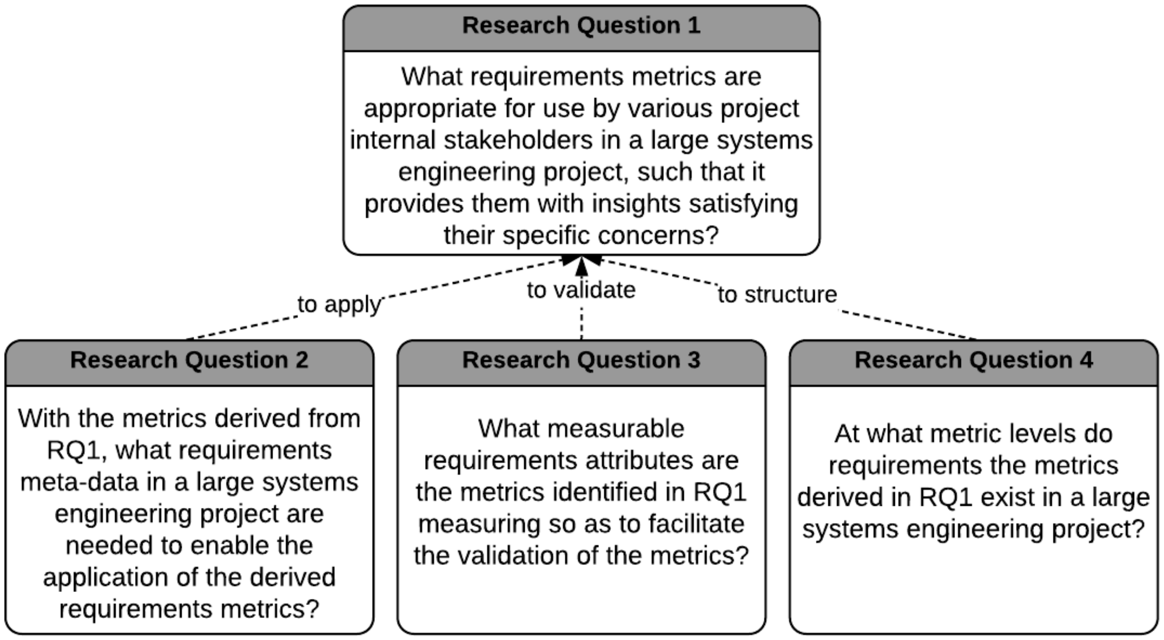

Research Question 1
What requirements metrics are appropriate for use by various project internal stakeholders in a large systems engineering project, such that it provides them with insights satisfying their specific concerns?
to apply
to validate
to structure
Research Question 2
With the metrics derived from RQ1, what requirements meta-data in a large systems engineering project are needed to enable the application of the derived requirements metrics?
Research Question 3
What measurable requirements attributes are the metrics identified in RQ1 measuring so as to facilitate the validation of the metrics?
Research Question 4
At what metric levels do requirements the metrics derived in RQ1 exist in a large systems engineering project?

Figure 3.2: Action research study research questions.

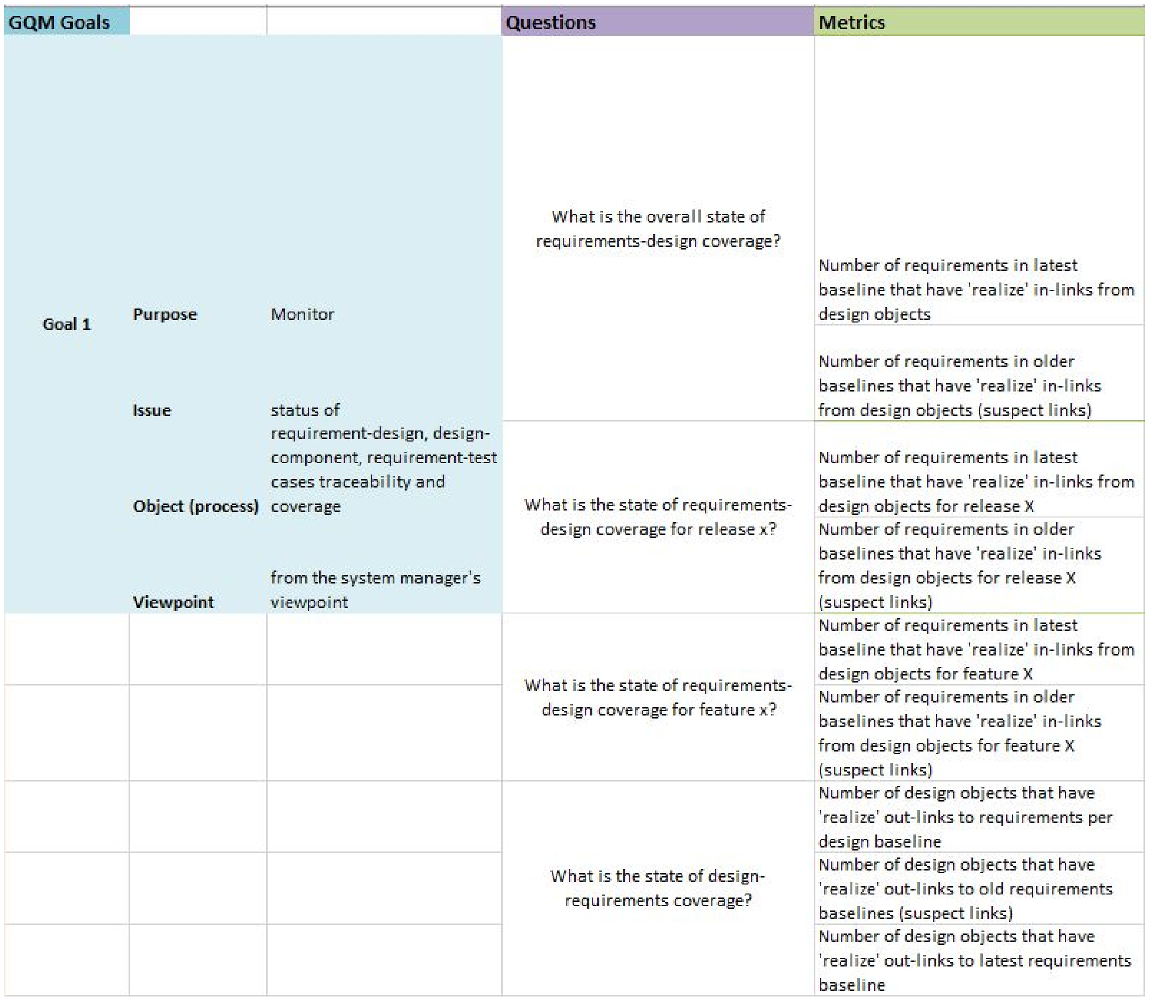
Phase 3: Intervention/Action Taking

To begin the metric definition process (i.e., solution design), the researcher used GQM [Basili et al., 1994] to create a list of high-level descriptions of the requirements metrics that are to be then further defined in detail. Figure 3.3 shows examples of such high-level metric descriptions. The list was then validated by the industrial participant. Based on the initial metric definitions, the researcher conducted an in-depth document analysis of the requirements, design, and test documents by exporting and gathering the data into spreadsheets and ensuring the completeness and consistency of the meta-data. Table 3.2 includes the number of requirement and design baselines (i.e., document versions), requirements, design objects, and test cases for each product that was included in the analysis. The researcher then used the gathered meta-data and high-level metric descriptions to define the metrics and calculate the measures using spreadsheets. Equation 3.1 is an example of a metric definition (all metric definitions are included and explained in detail in Section 3.4.4).

$$M1 : BaselineSize = \sum_{j=1}^{n} IsCORE(ReqID) \tag{3.1}$$

Given the full-time presence of the researcher at the project site, she continuously sought feedback on the metric definitions from the primary industrial participant until an initial set of metrics have been defined. Visualizations of the measures were created using excel.

To insert the designed solution (i.e., metrics) into the requirements management pro-



GQM Goals			Questions	Metrics
Goal 1	Purpose	Monitor	What is the overall state of requirements-design coverage?	Number of requirements in latest baseline that have 'realize' in-links from design objects
	Issue	status of requirement-design, design-component, requirement-test cases traceability and coverage		Number of requirements in older baselines that have 'realize' in-links from design objects (suspect links)
	Object (process)		What is the state of requirements-design coverage for release x?	Number of requirements in latest baseline that have 'realize' in-links from design objects for release X
	Viewpoint	from the system manager's viewpoint		Number of requirements in older baselines that have 'realize' in-links from design objects for release X (suspect links)
			What is the state of requirements-design coverage for feature x?	Number of requirements in latest baseline that have 'realize' in-links from design objects for feature X
				Number of requirements in older baselines that have 'realize' in-links from design objects for feature X (suspect links)
			What is the state of design-requirements coverage?	Number of design objects that have 'realize' out-links to requirements per design baseline
				Number of design objects that have 'realize' out-links to old requirements baselines (suspect links)
				Number of design objects that have 'realize' out-links to latest requirements baseline

Figure 3.3: An excerpt from the GQM template used in the AR study to derive the metrics.

cess and to gather feedback from internal stakeholders, two iterations of focus groups were held. In the first iteration, the measurements of Product 3 (see Table 3.2) were presented to a subset of the secondary industrial participants (i.e., internal stakeholders) which included the systems manager, R&D product manager, test mangers, two developers, architects, and the researcher. Feedback was gathered from the focus group and used to update the metrics. The feedback included suggestions for new metrics and enhancements to the visualizations. The second focus group included a larger group of secondary participants such as: program manager, quality managers, safety managers, and financial managers. The second focus group resulted in further feedback that was incorporated into the solution by the researcher. A second round of focus groups (two focus groups) was conducted the following month. The two rounds of focus groups familiarized the metrics to the internal project stakeholders. After the two round of focus groups, the researcher provided the up-to-date measurements to the stakeholders individually and upon request. Thus, the researcher received continuous feedback through direct engagement with the internal stakeholders and observation of the stakeholders' use of the measurements. Thus, the focus groups, observations and interactions with the internal stakeholder allowed for preliminary empirical validation of the metrics (see Section 3.5.1 for details on metric validation).

Phase 4: Evaluation/Analysis

The researcher conducted the evaluation and analysis of the intervention effects (i.e., inserting the derived metric into the requirements management and system development processes) through informal discussions with the primary and secondary industrial participants and observations of the processes. Issues such as improved requirement-design coverage, improved planning of time and effort per release, easier accessibility to the requirements information by internal stakeholders were noted (details on the empirical and theoretical validation of the metrics are in Section 3.5.1).

Moreover, the discussions with the internal stakeholders and observations of the processes led to further metric definition in order to address further internal stakeholders' concerns. For example, one coverage metrics is the *number of requirements with 'in-links' from design per baseline*. Upon discussions with the internal stakeholders, it became evident that there is a need for the *percentage of requirements with 'in-links' from design per baseline* and the metric was accordingly incorporated into the overall set of requirements metrics and its use validated by providing it to the internal stakeholders with the updated batch of measurements.

Phase 5: Reflections and Learning

The reflections and learning phase of an AR study aims to identify findings and lessons learned from the application of the proposed solution (i.e., metrics) and define further research questions that may emerge from the analysis of applying the metrics [Santos and Travassos, 2011]. In this phase, we began applying a measurement validation framework [Kitchenham, 1995] and structuring the measures into logical categories. Thus, given the iterative nature of an AR study, the need for research questions 3 and 4 emerged in the reflection and learning phase, which were then added to research questions 1 and 2. The findings from research questions 3 and 4 were then incorporated into the solution and evaluated as will be discussed in detail in Section 3.5.

3.4 Results

In this section, we describe the results from our AR study. The results are organized according to their associated research questions: measurable requirements attributes (RQ3), requirements metric levels (RQ4), requirements meta-data (RQ2), and requirements metrics (RQ1). Table 3.4 summarizes the AR results and maps them to their respective research questions (see Section 3.3.2 for research questions). We note that we report the results in a bottom-up approach; first we report the results of RQ3 (requirements attributes) and RQ4 (requirements metric levels) before the results of RQ2 (meta-data) and RQ1 (requirements metrics) to allow us to formulate the requirements metric definitions in light of the attributes and levels using the defined meta-data items. More specifically, in order for us to describe, for example, the metrics for requirements size (i.e., attribute) on the baseline level (i.e., metric level), one must have an understanding of *requirements size* and the *baseline level* first. Thus, our choice to report the findings of RQ3 and RQ4 first.

Table 3.4: Summary of AR study results.

Research Question	**Result Topic**	**Results**
RQ3	Requirements attributes	Size, growth, volatility, status, coverage
RQ4	Requirements metric levels	Baseline, feature, level, safety
RQ2	Requirements meta-data	Requirement ID, Requirement type, Feature ID, Release, Requirement text, Requirements status, safety requirement, Out-links, In-links.
RQ1	Requirements metrics	90 metrics

3.4.1 Measurable Requirements Attributes (RQ3)

The application of the measurement validation framework to the derived metrics in the reflection and learning phase of the AR study (see Section 3.3.2 for phase details) led to the identification of five requirements attributes that the derived metrics were measuring: size, growth, volatility, status, and coverage. We describe each attribute in detail in the following subsections.

Size

Size is the overall size of requirements to be implemented, designed, and tested within a given project or product. Wiegers [Wiegers, 2006] refers to it as product size and defines it as *the number of requirements in a body of work*. For example, we can say project A is larger than project B in terms of requirements size because project A has a larger number of requirements than project B. This is not to be confused with the size of individual requirements, a measure of requirement quality, which has received more attention than overall requirements size [Trudel and Abran, 2008; Génova et al., 2013; Antinyan and Staron, 2017]. This may be due to the diversity of requirement formats; requirements can be defined in terms of uses cases, functional requirements, business rules, etc. which can make it difficult to measure the overall requirements size. However, bringing overall requirements size into the limelight would encourage practitioners to set a definition within their projects as to what constitutes a requirement (see Section 3.4.4 for details on requirements size metrics). Moreover, measuring requirements size has implications for cost and effort estimation, negotiating contracts with the customers, and release planning.

Growth

Growth is the increase or decrease in requirements size between requirements baselines. For example, we can say the requirements growth between Requirement Baseline A and Requirement Baseline B is 50 requirements or 20%. We can also say the requirements growth between Baseline B and Baseline C is -10 or -5% because the requirements size decreased between baselines B and C (see Section 3.4.4 for details on requirements growth metrics). The literature is muddled on the definition and use of the term ‘requirements growth’. For one thing, requirements creep [Jones, 1996a; Kulk and Verhoef, 2008] and requirements scrap [Kulk and Verhoef, 2008] are other terms that have been used to describe the increase and decrease in overall requirements size. In addition, requirements growth is usually consumed under requirements volatility or change [Ebert and Dumke, 2007; Kulk and Verhoef, 2008]. We choose to use requirements growth as it encapsulates requirements creep and scrap but is distinct from volatility (see the following subsection for details on volatility). Measuring requirements growth allows for monitoring unwanted requirements growth and managing project resources in response to necessary requirements growth. For example, the detection of unanticipated growth of a requirements baseline that has been scheduled for release may trigger renegotiations of contracts, staffing, and pricing within the project and with customers.

Volatility

Volatility is the *tendency of requirements to change over time in response to the evolving needs of customers, stakeholders, organization, and work environment* [Nurmuliani et al., 2004]. More specifically, it is the number of additions, deletions and modifications made to a specific set of requirements over time [Costello and Liu, 1995]. The number of added, deleted, and modified requirements is determined in relation to another, usually older, set of requirements. For example, requirements volatility of Baseline A in relation to Baseline B is larger than requirements volatility of Baseline C in relation to Baseline D because the number of added, deleted and modified requirements between baseline A and B is greater than the number of added, deleted, and modified requirements between baselines C and D.

Requirements volatility has been studied widely in the literature [Zowghi and Nurmuliani, 2002; Thakurta and Ahlemann, 2010] with many metrics proposed to measure it ranging from simple [Wiegers, 2006] to more complex ones [Kulk and Verhoef, 2008]. It is also important to note that requirements volatility is different than requirements growth. One would argue that the number of added requirements would also indicate requirements growth. However, on the one hand, added requirements are concerned with the requirements that were not present in an older requirements baseline and then added in a later baseline and therefore is always a positive value. On the other hand, growth indicates the change in overall size, which takes into account both additions and deletions and may have a negative value. Measuring requirements volatility is important for managing development and maintenance effort, negotiating project scope, and rescheduling project deadlines [Thakurta and Ahlemann, 2010], among others.

Status

Status indicates the overall status of the project from requirements perspective, which is acquired by setting requirements to a *status* from a set of predefined states such as proposed, approved, implemented, and tested [Wiegers, 2006; Ebert and Dumke, 2007]. For example, the status of Baseline A is 50% approved, 40% implemented, and 10% tested. It is good requirements engineering practice to allocate a status to requirements and track the number and evolution of requirement statuses over time [Wiegers, 2006; Ebert and Dumke, 2007]. Hence, measuring requirements status aids in avoiding the pervasive “90% done” problem [Wiegers, 2006] by allowing internal stakeholders to provide more accurate measures of their progress with regard to the requirements. However, the literature lacks in-depth investigations of requirements status as a measurable requirement attribute.

Coverage

Coverage is the degree to which a set of requirements are covered by other software artifacts in the development lifecycle, particularly architecture, development, and testing artifacts. In other words, it is the degree to which a set of requirements have identifiable links to other software artifacts. Therefore, we can say Baseline B has higher requirements-to-design coverage (see Section 3.4.4 for details on requirements coverage metrics) than Baseline A because Baseline B has a larger number of requirements that have links to design artifacts than Baseline A. Requirements coverage is closely linked to but different from the extensively studied field of requirements traceability, although the literature sometimes refers to requirements coverage as 'traceability' [Nassar et al., 2016]. While traceability is *the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)* [Gotel and Finkelstein, 1994], we use the term coverage to indicate the measurements derived from traceability data (e.g., links to design, test cases, etc.). Therefore, in order to obtain precise coverage measures for a set of requirement, good requirements traceability is necessary. Moreover, coverage is usually used to refer to test coverage of the requirements, which has received the bulk of research efforts. However, our experience with systems engineering projects shows that requirements-design coverage is also an important aspect of coverage. Measuring requirements coverage would aid in improving requirements traceability and tracking project progress with respect to design, implementation, and testing.

3.4.2 Requirements Metric Levels (RQ4)

Upon identifying the requirements attributes measured by the derived metrics, we addressed the issue of unstructured and unorganized metrics by grouping them into related clusters. We discovered that the clusters, which we refer to as metric levels, are based on the different ways that requirements were organized in our project. At the highest level, requirements were organized into baselines. The requirements within the baselines can then be organized according to features, releases, and safety requirements. Thus, we say: baseline size metrics, feature size metrics, release size metrics, safety requirements size metrics and so on for the rest of the attributes (i.e., growth, volatility, status, and coverage). We describe each requirement metric level in the following subsections.

Baseline Level

A requirements baseline consists of all the specified requirements for a given product until that point in time. For example, we say we are measuring baseline size or requirements size at the baseline level through a set of defined metrics (see Section 3.4.4 for baseline size metric details). In Figure 3.4, the first figure at the top left shows that Baseline 3.1 consists of *n* requirements. Thus, according to the defined baseline size metric, the size of baseline

3.1 is *n* requirements. The same applies for growth, volatility, status and coverage.

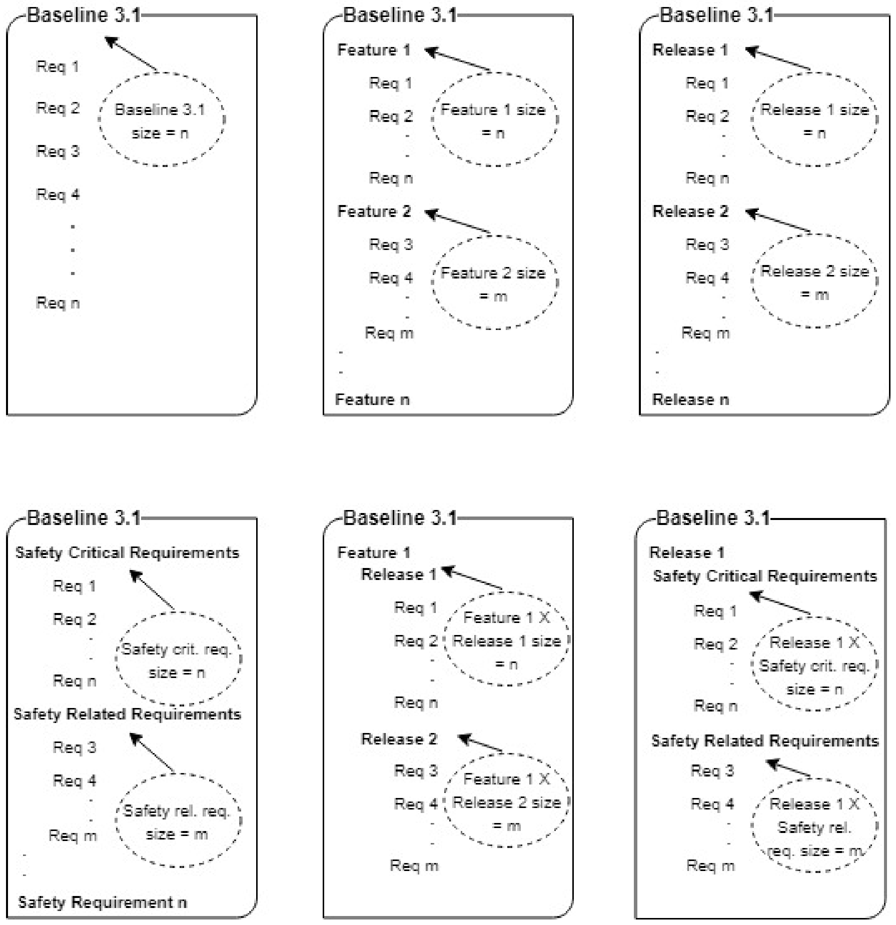


Figure 3.4: Examples of requirements metric levels.

Feature Level

A feature is a unit of functionality of a software system that is represented by a set of requirements within a requirements baseline. The entire set of features for a product will constitute the requirements baseline. In other words, the requirements baseline consists of requirements that are grouped into features. For example, we say we are measuring feature size or requirements size at the feature level through a set of defined metrics (see Section 3.4.4 for feature size metric details). In Figure 3.4, the second figure at the top row shows Baseline 3.1 organized into Feature 1, Feature 2, Feature *n*. Feature 1 consists of *n* requirements and feature 2 consists of *m* requirements. Thus, according to the defined feature size metric, the size of Feature 1 in Baseline 3.1 is *n* requirements and the size of Feature 2 in Baseline 3.1 is *m* requirements and so on until Feature *n*. The same applies for growth, volatility, status and coverage

Release Level

A software release consists of a set of requirements that the client has agreed upon to be developed and delivered to them by a specific date. Requirements in a baseline are organized according to a number of predefined releases. For example, we say we are measuring release size or requirements size at the release level through a set of defined metrics (see Section 3.4.4 for release size metric details). In Figure 3.4, the third figure at the top row shows Baseline 3.1 organized into Release 1, Release 2, Release *n*. Release 1 consists of *n* requirements and Release 2 consists of *m* requirements. Thus, according to the defined release size metric, the size of Release 1 in Baseline 3.1 is *n* requirements and the size of Release 2 in Baseline 3.1 is *m* requirements and so on until Release *n*. The same applies for growth, volatility, status and coverage.

Safety Level

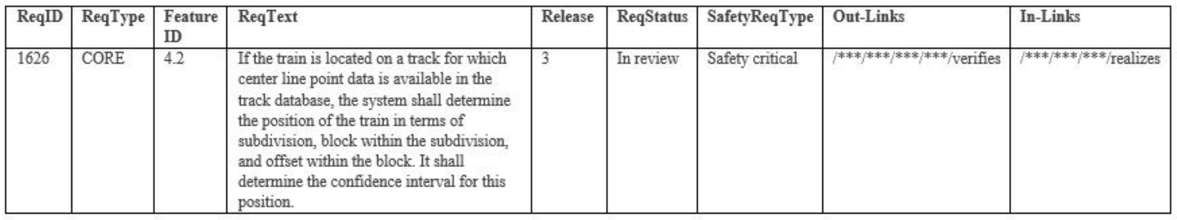
Systems projects often consist of safety critical components. Thus, it comes as no surprise that there was a need to define metrics for safety requirements and that when we organized the defined metrics, safety requirements constituted a cluster of the defined metrics. Requirements in a baseline can be organized according to their safety relevance. For example, we say we are measuring safety requirements size or requirements size at the safety level through a set of defined metrics (see Section 3.4.4 for safety requirements size metric details). In Figure 3.4, the first figure in the lower row shows Baseline 3.1 organized into Safety Critical requirements and Safety Relevant requirements. Safety Critical requirements consist of *n* requirements and Safety Critical requirements consist of *m* requirements. Thus, according to the defined safety requirements size metric, the size of Safety Critical requirements in Baseline 3.1 is *n* requirements and the size of Safety Relevant requirements in Baseline 3.1 is *m* requirements. The same applies for growth, volatility, status and coverage.

Interestingly, we can also combine levels and derive metrics for different level combinations. For example, the second figure in the second row shows that requirements can be organized into features and, within the features, requirements can be organized according to release. Thus, we can measure the size of Release 1 for Feature 1 in Baseline 3.1 through a set of defined metrics. The same applies to the last figure in the second row where requirements in a baseline are organized according to release and the requirements within a release are organized according to safety relevancy. Thus, we would need metrics to measure the size of safety critical requirements for Release 1 in Baseline 3.1. Such level combinations allow for deriving further metrics at lower levels if the need for them arises.

3.4.3 Requirements Meta-Data (RQ2)

In Phase 3 of the AR study (see Section 3.3.2) we gathered the requirements meta-data for the products under study (see Table 3.2) in spreadsheets and, upon defining an initial set of requirements metrics, the need for further meta-data items emerged to define the remaining metrics. Thus, we re-exported the requirements baselines with the complete and correct requirements meta-data. To avoid such unnecessary rework, a complete and consistent list of meta-data is necessary for defining the metrics and applying the metrics in this paper. In this subsection, we define the 9 requirements meta-data items that will be used in the metrics in Section 3.4.4 below. Table 3.5 includes the list of meta-data items, their short names that will be used in the metric definitions below, and each meta-data item's description. Figure 3.5 shows an excerpt from our project data with an example for each of the meta-data items in Table 3.5.

The requirements meta-data items are not limited to items listed in Table 3.5. The projects under study consisted of other meta-data items that we did not include in this paper such as requirement author, last modification date, and requirement history, to name a few, which served different purposes within the project. However, the identified list consists of the meta-data items that are necessary for the metrics defined in this paper.



ReqID	ReqType	Feature ID	ReqText	Release	ReqStatus	SafetyReqType	Out-Links	In-Links
1626	CORE	4.2	If the train is located on a track for which center line point data is available in the track database, the system shall determine the position of the train in terms of subdivision, block within the subdivision, and offset within the block. It shall determine the confidence interval for this position.	3	In review	Safety critical	/***/***/***/***/verifies	/***/***/***/realizes

Figure 3.5: Examples of requirements meta-data.

On Meta-Data Values. Given the centrality of the identified meta-data items in the following metric definitions, it is important to note that the accuracy of the resultant measures is dependent on the quality of meta-data values. In other words, the metrics require *sanitized* data as input in order to calculate accurate measures. For example, if we are to calculate the number of requirements that have in-links from design (i.e., requirements-design coverage), then the *in-links* meta-data values (see Figure 3.5) must be complete and correct for *all* requirements in order to calculate an accurate requirements-design coverage measure. Thus, if the internal stakeholder responsible for maintaining such links erred and did not establish a link between artifacts or if a human incorrectly created a link, the metrics will not be able to detect it.

Table 3.5: Requirements meta-data required to apply the metrics.

	Requirement Meta-Data Item	**Meta-Data Item Short Name**	**Description**
1	Requirement ID	ReqID	A unique identification number for each requirement in the database automatically generated by DOORS. If the requirements is deleted, the ID number will not be reassigned to another requirement.
2	Requirement type	ReqType	Each requirements baseline in the DOORS database consists of large number of rows. The rows consist of diagrams, descriptive texts, and requirements. Requirements that are to be designed, implemented, tested and delivered need to be assigned a type. We identify such requirements as CORE. Thus, the metrics apply only to the CORE requirements.
3	Requirement feature ID	FeatureID	Requirements are organized into features and sub-features and each is assigned a feature ID. For example, external interface requirements are a high-level feature (feature ID = 1) that is further organized into communication interface requirements (feature ID = 1.1), user interface requirements (feature ID = 1.2), and maintenance interface requirements (feature ID = 1.3).
4	Requirement text	ReqText	Consists of the requirement text in natural language.
5	Requirement release number	Release	Each requirement is assigned a release number as agreed upon by the internal stakeholder in charge of release management.
6	Requirement status	ReqStatus	Each requirement is assigned a status such as 'approved', 'in review', 'implemented', and 'tested'. The requirements states should be agreed upon by the internal stakeholders and incorporated into the requirements management plan to ensure consistency.
7	Safety requirement type	Safety	Many systems projects are in a safety critical domain, such as the rail-automation domain of our AR study. Thus, each requirement is assigned a safety level. In the project under study, the safety levels were: regular, safety relevant, and safety critical.
8	Out-links from requirements to external artifacts	OutLinks	Contains outgoing links from requirements to external artifacts such as tests and implementation.
9	In-Links to requirements from external artifacts	InLinks	Contains incoming links from external artifacts such as design to requirements. We note that the links to and from requirements depend on the tracing model adopted by the particular project.

3.4.4 Requirements Metrics (RQ1)

In this subsection, we discuss the requirements size, growth, volatility, status, and coverage metrics we defined. Specifically, we report the metrics for each measurable attribute on the baseline, feature, release and safety levels. In other words, if A is the set of requirements attributes such that: $A = \{Size, Growth, Volatility, Status, Coverage\}$, and L the set of requirements levels such that: $L = \{Baseline, Feature, Release, Safety\}$, then the metrics are organized according to each pair in the $A \times L$ set:

$$\begin{aligned}
A \times L = \{&(Size, Baseline), (Size, Feature), (Size, Release), (Size, Safety),\\
&(Growth, Baseline), (Growth, Feature), (Growth, Release), (Growth, Safety),\\
&(Volatility, Baseline), (Volatility, Feature), (Volatility, Release), (Volatility, Safety),\\
&(Status, Baseline), (Status, Feature), (Status, Release), (Status, Safety),\\
&(Coverage, Baseline), (Coverage, Feature), (Coverage, Release), (Coverage, Safety)\}
\end{aligned}$$

Table 3.6 provides a summary of the metrics for each $A \times L$ pair. Then, in the following subsections (denoted by $Attribute \times Level$), we define the metrics and give examples of the measures and their visualizations based on data from the three products in Table 3.2. For simplicity, we refer to the product in the examples within this section as *Product X*.

Table 3.6: Summary of defined requirements metrics.

Requirement Attribute	**Requirement Metric Level**	**Number of Metrics**	**Metrics**
Size	Baseline	1	M1
	Feature	2	M2, M3
	Release	2	M4, M5
	Release X Feature	3	M6, M7, M8
	Safety	2	M9
Growth	Baseline	2	M11, M12
	Feature	2	M13, M14
	Release	2	M15, M16
	Safety	2	M17, M18
Volatility	Baseline	8	M19, M20, M21, M22, M23, M24, M25, M26
	Feature	8	M27, M28, M29, M30, M31, M32, M33, M34
	Release	8	M35, M36, M37, M38, M39, M40, M41, M42
	Safety	8	M43, M44, M45, M46, M47, M48, M49, M50
Status	Baseline	2	M51, M52
	Feature	2	M53, M54
	Release	2	M55, M56
	Safety	2	M57, M58
Coverage	Baseline	8	M59, M60, M61, M62, M63, M64, M65, M66
	Feature	8	M67, M68, M69, M70, M71, M72, M73, M74
	Release	8	M75, M76, M77, M78, M79, M80, M81, M82
	Safety	8	M83, M84, M85, M86, M87, M88, M89, M90
		Total: 90	

Size Metrics

We defined 10 metrics for requirements size on the baseline, feature, release, and safety levels, which we describe below.

Size × Baseline Level On the baseline level, the size metric is a count of requirements in a given baseline. As mentioned in Section 3.4.3, we are interested in the *CORE* requirements only. Thus, Equation 3.2 tests whether a requirement is a *CORE* requirement. Equation 3.3 (M1) counts the number of *CORE* requirements in baseline *i*.

$$isCORE(ReqID) = \begin{cases} 1, & ReqType = CORE \\ 0, & ReqType \neq CORE \end{cases} \tag{3.2}$$

$$M1 : BaselineSize_i = \sum_{j=1}^{n} IsCORE(ReqID_j) \tag{3.3}$$

Where:
i is baseline number for which size is being calculated
j is the requirement count in the baseline
n is the number of requirements in the baseline
ReqID is requirement unique ID in baseline

Table 3.7 consists of baseline sizes for one of the products in Table 3.2 and Figure 3.6 depicts the baseline size measures in a bar graph.

Baseline	M1: BaselineSize
1.0	755
1.1	755
1.2	743
1.3	754
1.4	784
1.5	779
1.6	895
1.7	892
2.0	884

Table 3.7: Baselines sizes for Product X.

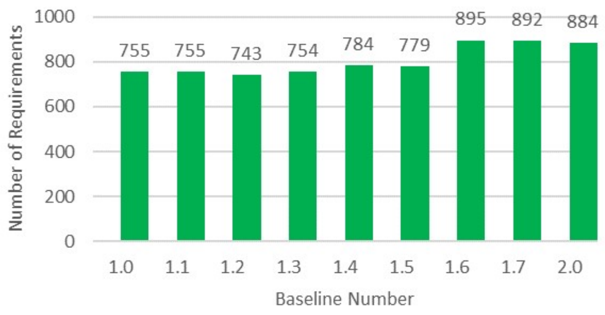


Figure 3.6: Baseline sizes for Product X.

Size × Feature Level Equation 3.4 tests whether a requirement is a *CORE* requirement and whether its *feature ID* matches the given feature ID for which size is being calculated. Equation 3.5 counts the number of *CORE* requirements in feature *i*. While Equation 3.5 (M2) results in an absolute number, Equation 3.6 (M3) below calculates the feature size percentage in relation to the baseline size.

$$isCOREandFeature(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge FeatureID = i \\ 0, & otherwise \end{cases} \tag{3.4}$$

$$M2: FeatureSizeA_i = \sum_{j=1}^{n} IsCOREandFeature(ReqID_j, i) \tag{3.5}$$

$$M3: FeatureSizeP_i = \frac{FeatureSizeA_i}{BaselineSize} \times 100 \tag{3.6}$$

Where:
i is feature ID for which size is being calculated
j is the requirement count in feature
n is the number of requirements in the feature
ReqID is requirement unique ID in baseline

Table 3.8 shows feature size (absolute (M2) and percentage (M3)) for features 4.1 – 4.10 (feature names have been omitted for privacy reasons) in baseline 4.4 for Product X. Figure 3.7 depicts the feature size measures in a bar graph. We can see that feature 4.8 constitutes almost 10% of overall baseline and consists of 72 *CORE* requirements, which indicates a big feature. The measure can be visualized in different ways for different audiences. For example, the complete list of feature within a baseline can be depicted in a pie chart to depict their sizes in relation to feature and overall baseline size.

Table 3.8: Feature sizes in baseline 4.4 for Product X.

Feature	**M2: FeatureSizeA**	**M3: FeatureSizeP**
4.1	9	1.19%
4.2	31	4.11%
4.3	54	7.15%
4.4	4	0.53%
4.5	2	0.26%
4.6	10	1.32%
4.7	23	3.05%
4.8	72	9.54%
4.9	47	6.23%
4.10	5	0.66%

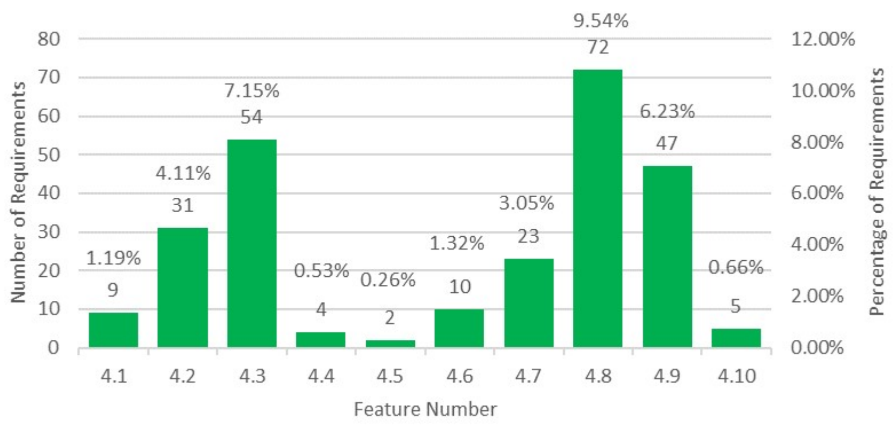


Figure 3.7: Feature size in baseline 4.4 for Product X.

Size × Release Level Equation 3.7 tests whether a requirement is a *CORE* requirement and whether its release number matches the given release number for which size is being calculated. Equation 3.8 (M4) counts the number of *CORE* requirements in release *i*. Equation 3.9 (M5) calculates the release size percentage in relation to the baseline size.

$$isCOREandRelease(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge Release = i \\ 0, & otherwise \end{cases} \tag{3.7}$$

$$M4 : ReleaseSizeA_i = \sum_{j=1}^{n} IsCOREandRelease(ReqID_j, i) \tag{3.8}$$

$$M5 : ReleaseSizeP_i = \frac{ReleaseSizeA_i}{BaselineSize} \times 100 \tag{3.9}$$

Where:
i is release number for which size is being calculated
j is the requirement count in release
n is the number of requirements in the release
ReqID is requirement unique ID in baseline

Table 3.9 consists of release size measures (absolute (M4) and percentage (M5)) for releases 1-7 in baseline 4.4 for Product X. Figures 3.9 and 3.8 show alternative ways release size can be visualized. We notice that release 7 is the largest in size constituting 28% of the overall requirements (i.e., baseline) size and consisting of 675 requirements to be design, implemented, tested, and delivered.

Table 3.9: Release sizes in Baseline 4.4 for Product X.

Feature	**M4: ReleaseSizeA**	**M5: ReleaseSizeP**
1	179	7.49%
2	470	19.67%
3	225	9.42%
4	252	10.55%
5	341	14.27%
6	240	10.05%
7	675	28.25%

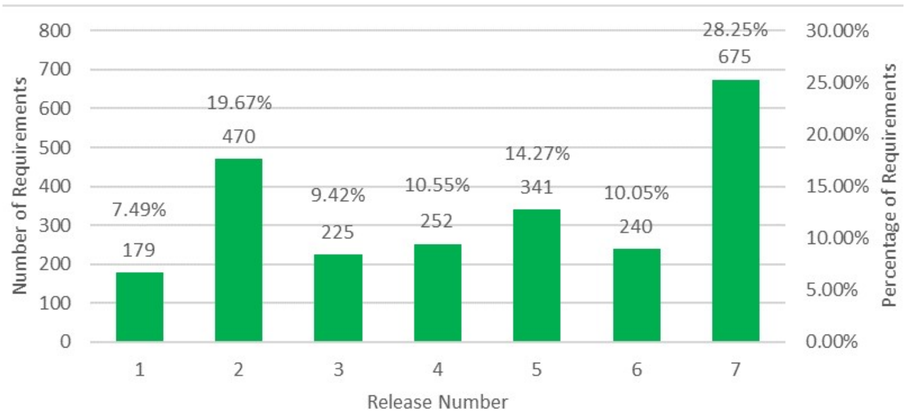


Figure 3.8: Release sizes in Baseline 4.4 for Product X in bar graph visualization.

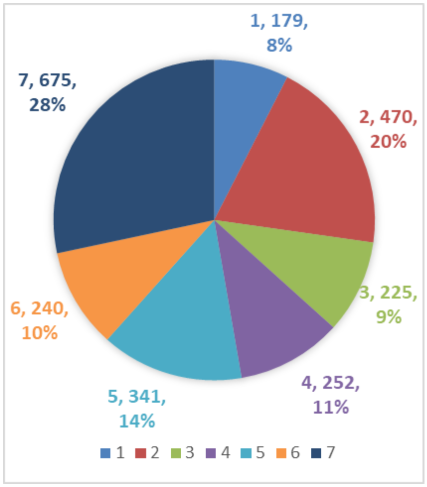


Figure 3.9: Release sizes in Baseline 4.4 for Product X in pie chart visualization.

Size × Release × Feature Level There was need to learn how many requirements are within a given release and feature. In other words, one would ask: how many requirements for the *start up* feature are allocated to *release 7*. To address this need, we derived three metrics: i) the number of requirements per feature per release per baseline (M6) represented in Equation 3.11, ii) the percentage of requirements per feature per release per baseline in relation to feature size (M7) represented in Equation 3.12, and iii) the percentage of requirements per feature per release per baseline in relation to release size (M8) represented in Equation 3.13.

M6 is an intersection of the requirements in feature *i* and release *j* for any given baseline, which results in an absolute value. However, M7 and M8 warrant a more detailed explanation. For M7, the number of requirements in feature *i* and release *j* in a given baseline is divided by feature *i* size. This allows us to know the percentage of requirements for feature *i* that have been allocated to release *j*. For example, we can say: 10% of the requirements of feature *i* are allocated to *release 7*. On the other hand, in M8, the number of requirements in feature *i* and release *j* in a given baseline is divided by release *j* size. This allows us to know the percentage of release *j* requirements that are for feature *i*. For example, we can say: 10% of release 1 is feature a, 30% of release 1 is of feature b, etc.

$$isCOREandFeatureandRelease(ReqID, i, j) = \begin{cases} 1, & ReqType = CORE \wedge FeatureID = i \wedge Release = j \\ 0, & otherwise \end{cases} \tag{3.10}$$

$$M6: ReleaseFeatureSizeA_i = \sum_{w=1}^{n} IsCOREandFeatureandRelease(ReqID_w, i, j) \tag{3.11}$$

$$M7: ReleaseFeatureSizeP1_{ij} = \frac{ReleaseFeatureSizeA_{ij}}{FeatureSize_i} \times 100 \tag{3.12}$$

$$M8: ReleaseFeatureSizeP2_{ij} = \frac{ReleaseFeatureSizeA_{ij}}{ReleaseSize_j} \times 100 \tag{3.13}$$

Where:
i is feature ID for which size is being calculated
j is release number for which size is being calculated
w is the requirement count in release and feature
n is the number of requirements in the release and feature
ReqID is requirement unique ID in baseline

To demonstrate the use of metrics M6, M7, and M8, we shall give two examples from our project data. The first example shows Feature X Release sizes of ten features for release 3.

That is, the number of requirements in features 4.1–4.10 that have been allocated to Release 3. The second example shows Feature X Release sizes of eight releases for feature 4.3. That is, the number of requirements from Feature 4.3 allocated to releases 1–8.

Table 3.10 shows the sizes of features 4.1-4.10 for release 3. We see that the size of features 4.1, 4.5, 4.7, 4.8, and 4.9 are zero in release 3 meaning that no requirements from these features have been allocated to release 3. M7 in Table 3.10 shows that 100% of features 4.2, 4.4, 4.6 have been allocated to release 3 while 41.30% of feature 4.3 and 29.03% of feature 4.10 has been allocated to release 3. On the other hand, M8 in Table 3.10 shows that Release 3 is 19.11% feature 4.2, 16.89% feature 4.3, 5.33% feature 4.4, 1.33% feature 4.6, and 4% feature 4.10. Thus, M8 provides us with the constituents of release 3. Figure 3.10 shows the constituents of release 3 in a pie chart.

Table 3.10: Feature sizes in Release 3 in Baseline 4.4 for Product X.

Feature	**M2: FeatureSizeA**	**Release**	**M4: ReleaseSizeA**	**M6: FeatureRelease SizeA**	**M7: FeatureSizeP1**	**M8: FeatureSizeP2**
4.1	6	3	225	0	0.00%	0.00%
4.2	43	3	225	43	100.0%	19.11%
4.3	92	3	225	38	41.30%	16.89%
4.4	12	3	225	12	100.00%	5.33%
4.5	2	3	225	0	0.00%	0.00%
4.6	3	3	225	3	100.00%	1.33%
4.7	195	3	225	0	0.00%	0.00%
4.8	212	3	225	0	0.00%	0.00%
4.9	188	3	225	0	0.00%	0.00%
4.10	31	3	225	9	29.03%	4.00%

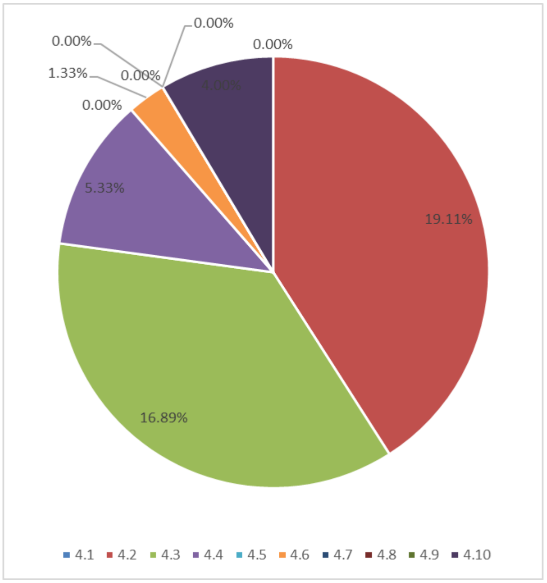


Figure 3.10: Feature sizes in Release 3 in Baseline 4.4 for Product X.

Table 3.11 shows the size of Feature 4.3 over releases 1–8. M7 shows that 41.30% of feature 4.3 is in release 3, 25% in release 4, 5.43% in release 5, 16.30% in release 6 and 11.96% in release 7. Thus, M7 gives us an idea about a feature's timeline. Figure 3.11 visualizes the timeline of feature 4.3 over releases 1–8 in a bar graph. The red line indicates the size of feature 4.3 (i.e., 92 requirements).

Table 3.11: Size of Feature 4.3 in relation to release in Baseline 4.4 for Product X.

Release	**M2: ReleaseSizeA**	**Feature**	**M4: FeatureSizeA**	**M6: ReleaseFeature SizeA**	**M7: ReleaseFeature SizeP1**	**M8: ReleaseFeature SizeP2**
1	179	4.3	92	0	0.00%	0.00%
2	470	4.3	92	0	100.0%	19.11%
3	225	4.3	92	38	41.30%	16.89%
4	252	4.3	92	23	25.00%	9.13%
5	341	4.3	92	5	5.43%	1.47%
6	240	4.3	92	15	16.30%	6.25%
7	675	4.3	92	11	11.96%	1.63%
8	7	4.3	92	0	0.00%	0.00%

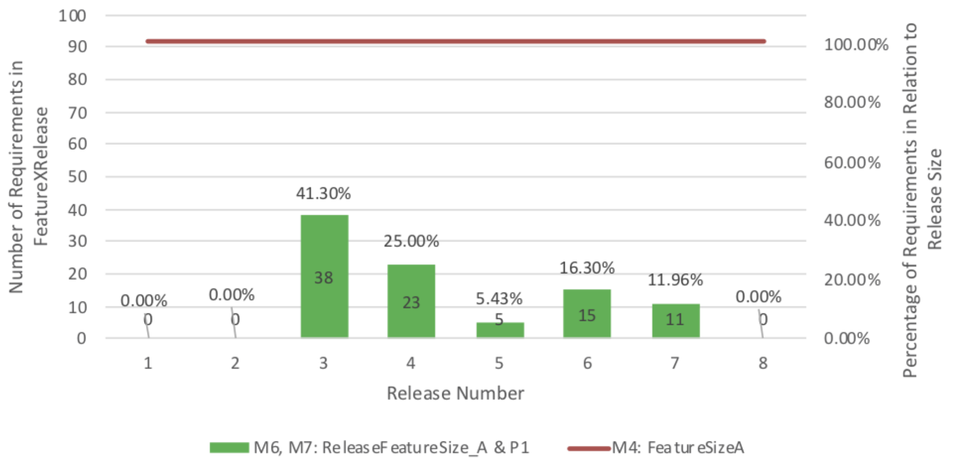


Figure 3.11: Size of Feature 4.3 in relation to release in Baseline 4.4 for Product X.

Size × **Safety Level** Equation 3.14 tests whether a requirement is a *CORE* requirement and whether its safety level matches the safety level for which size is being calculated. M9 counts the number of *CORE* requirements in safety level *i*. M10 calculates the safety level size percentage in relation to the baseline size. Table 3.12 shows an example of the use of M9 and M10 on a sample of our data.

$$isCOREandSafety(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge Safety = i \\ 0, & otherwise \end{cases} \tag{3.14}$$

$$M9 : SafetySizeA_i = \sum_{j=1}^{n} IsCOREandSafety(ReqID_j, i) \tag{3.15}$$

$$M10 : SafetySizeP_i = \frac{SafetySizeA_i}{BaselineSize} \times 100 \tag{3.16}$$

Where:
i is safety level for which size is being calculated
j is the requirement count in safety level
n is the number of requirements in safety level
ReqID is requirement unique ID in baseline

Table 3.12: Safety level sizes for Product X.

Safety Level	**M9: SafetySizeA**	**M10: SafetySizeP**
Regular safety	558	23.36%
Safety relevant	115	4.86%
Safety critical	487	20.30%

Growth Metrics

We defined 8 metrics on the baseline, feature, release, and safety levels.

Growth × **Baseline Level** Baseline growth metrics consist of two metrics: the difference between two requirements baseline sizes (an absolute value) (M11) represented by Equation 3.17 and the percentage of the difference between two requirements baseline sizes (M12) represented by Equation 3.18. Requirements growth measures can have either a negative or positive value. Figure 3.12 shows baseline growth for Product X for baselines 3.2 – 3.5.3. We see that there was large growth in baseline 3.5.1, which coincided with a major product release.

$$M11 : BaselineGrowthA_i = BaselineSizeA_i - BaselineSizeA_j \tag{3.17}$$

$$M12 : BaselineGrowthP_i = \frac{BaselineGrowthA_i}{BaselineSizeA_j} \times 100 \tag{3.18}$$

Where:
i is baseline number for which growth is being calculated
j is the baseline number for an older baseline

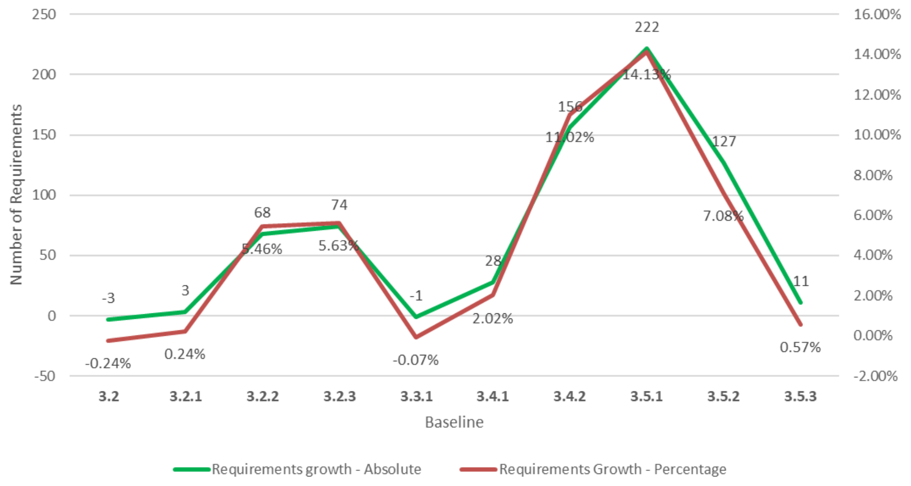


Figure 3.12: Baseline growth for Product X.

Growth × Feature Level We defined two feature growth metrics: the difference between feature size in two baselines resulting in an absolute value (M13) represented by Equation 3.19 and the percentage of the difference between feature size in two baselines resulting in a relative value (M14) represented by Equation 3.20. Figure 3.13 shows a sample graph of Feature 4.8 growth over baselines 3.2–3.5.2.

$$M13 : FeatureGrowthA_{wi} = FeatureSizeA_{wi} - FeatureSizeA_{wj} \tag{3.19}$$

$$M14 : FeatureGrowthP_{wi} = \frac{FeatureGrowthA_{wi}}{FeatureSizeA_{wj}} \times 100 \tag{3.20}$$

Where:
w is feature number for which growth is being calculated
i is baseline number in which feature growth is being calculated
j is the baseline number for an older baseline

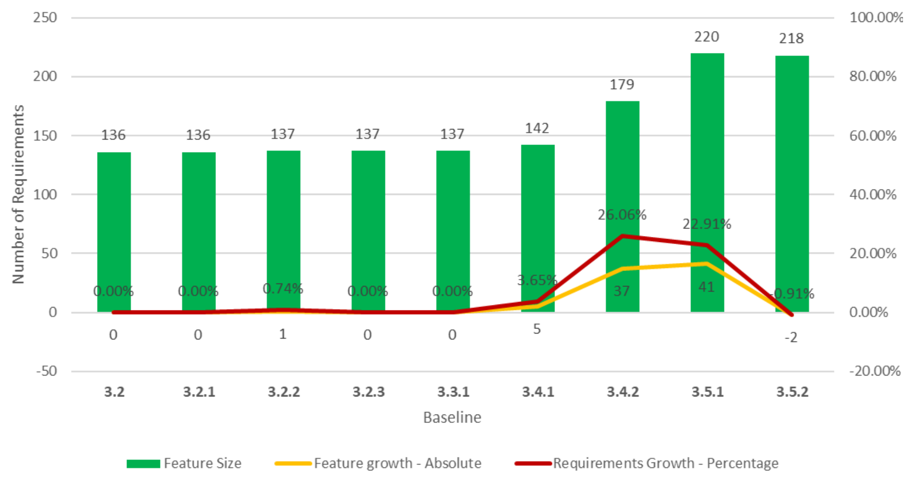


Figure 3.13: Feature 4.8 growth for Product X.

Growth × Release Level We defined two release growth metrics: the difference between feature size in two baselines (absolute value) (M15) represented by Equation 3.21 and the percentage of the difference between feature size in two baselines (relative value) (M16) represented by Equation 3.22.

$$M15: ReleaseGrowthA_{wi} = ReleaseSizeA_{wi} - ReleaseSizeA_{wj} \tag{3.21}$$

$$M16: ReleaseGrowthP_{wi} = \frac{ReleaseGrowthA_{wi}}{ReleaseSizeA_{wj}} \times 100 \tag{3.22}$$

Where:
w is release number for which growth is being calculated
i is baseline number in which release growth is being calculated
j is the baseline number for older baseline

Growth × Safety Level We defined two safety level growth metrics: the difference between safety level size in two baselines (absolute value) (M17) represented by Equation 3.23 and the percentage of the difference between safety level size in two baselines (relative value) (M18) represented by Equation 3.24. Release and safety level growth can be depicted similar to feature growth in Figure 3.13.

$$M17: SafetyGrowthA_{wi} = SafetySizeA_{wi} - SafetySizeA_{wj} \tag{3.23}$$

$$M18: SafetyGrowthP_{wi} = \frac{SafetyGrowthA_{wi}}{SafetySizeA_{wj}} \times 100 \tag{3.24}$$

Where:
w is safety level for which growth is being calculated
i is baseline number in which safety growth is being calculated
j is the baseline number for an older baseline

Volatility Metrics

We defined 32 volatility metrics on the baseline, feature, release, and safety levels.

Volatility × Baseline Level On the baseline level, we measure the number of additions, modifications, deletions, and total churn for requirements in a baseline. The number of *additions* is measured by counting the number of requirements that exist in the current baseline (i) but not in an older baseline (j). Equation 3.25 tests the conditions then Equation 3.26 (M19) counts the number of added requirements per baseline resulting in an absolute value. Equation 3.27 (M20) calculates the percentage of added requirements in relation to baseline size.

$$isAdded(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge ReqID \in Baseline_i \wedge ReqID \notin Baseline_j \\ 0, & otherwise \end{cases} \tag{3.25}$$

$$M19 : BaselineAddedA_i = \sum_{w=1}^{n} IsAdded(ReqID_w) \tag{3.26}$$

$$M20 : BaselineAddedP_i = \frac{BaselineAddedA_i}{BaselineSize_i} \times 100 \tag{3.27}$$

The number of *deletions* is measured by counting the number of requirements that are in an older baseline (j) but not in the current baseline (i). Equation 3.28 tests these conditions then Equation 3.29 (M21) counts the number of deleted requirements per baseline resulting in an absolute value. Equation 3.30 calculates the percentage of deleted requirements in relation to baseline size (M22).

$$isDeleted(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge ReqID \notin Baseline_i \wedge ReqID \in Baseline_j \\ 0, & otherwise \end{cases} \tag{3.28}$$

$$M21: BaselineDeletedA_i = \sum_{w=1}^{n} IsDeleted(ReqID_w) \tag{3.29}$$

$$M22: BaselineDeletedP_i = \frac{BaselineDeletedA_i}{BaselineSize_i} \times 100 \tag{3.30}$$

The number of *modifications* is measured by counting the number of requirements that are in an older baseline (j) and in the current baseline (i), but its text has been modified. Equation 3.31 tests these conditions then Equation 3.32 (M23) counts the number of modified requirements per baseline resulting in an absolute value. Equation 3.33 calculates the percentage of deleted requirements in relation to baseline size (M24).

$$isModified(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge ReqID \in Baseline_i \wedge ReqID \in Baseline_j \wedge ReqText_i \neq ReqText_j \\ 0, & otherwise \end{cases} \tag{3.31}$$

$$M23: BaselineModifiedA_i = \sum_{w=1}^{n} IsModified(ReqID_w) \tag{3.32}$$

$$M24: BaselineModifiedP_i = \frac{BaselineModifiedA_i}{BaselineSize_i} \times 100 \tag{3.33}$$

Finally, requirements *churn* on the baseline level is measured by adding the number of added, deleted, and modified requirements, which can be achieved by using Equation 3.34 (M25) resulting in an absolute value. Equation 3.35 calculates the percentage of requirements churn in relation to baseline size (M26).

$$\begin{gathered} M25: BaselineChurnA_i = \\ BaselineAddedA_i + BaselineDeletedA_i + BaselineModifiedA_i \end{gathered} \tag{3.34}$$

$$M26: BaselineChurnP_i = \frac{BaselineChurnA_i}{BaselineSize_i} \times 100 \tag{3.35}$$

Where:

i is baseline for which volatility is being calculated

j is an older baseline number

w is the requirement count in baseline

n is the number of requirements in baseline *i*

Table 3.13 shows sample baseline volatility measures for Product X from our data and Figure 3.14 visualizes the data. We excluded percentages (M20, M22, M24, M26) from the figure to maintain readability of the graph.

Table 3.13: Baseline volatility for Product X.

Baseline	M1: Baseline_ SizeA	M19: Baseline_ AddedA	M20: Baseline_ AddedP	M21: Baseline_ DeletedA	M22: Baseline_ DeletedP	M23: Baseline_ ModifiedA	M24: Baseline_ ModifiedP	M25: Baseline_ ChurnA	M26: Baseline_ ChurnP
3.2	1243	8	0.64%	11	0.88%	17	1.37%	36	2.90%
3.2.1	1246	3	0.24%	0	0.00%	13	1.04%	16	1.28%
3.2.2	1314	78	5.94%	10	0.76%	102	7.76%	190	14.46%
3.2.3	1388	86	6.20%	12	0.86%	106	7.64%	204	14.46%
3.3.1	1387	1	0.07%	2	0.14%	38	2.74%	41	2.96%
3.4.1	1415	64	4.52%	36	2.54%	81	5.72%	181	12.79%
3.4.2	1571	163	10.38%	7	0.45%	66	4.20%	236	15.02%
3.5.1	1793	323	18.01%	101	5.63%	107	5.97%	531	29.62%
3.5.2	1920	138	7.19%	11	0.57%	54	2.81%	203	10.57%

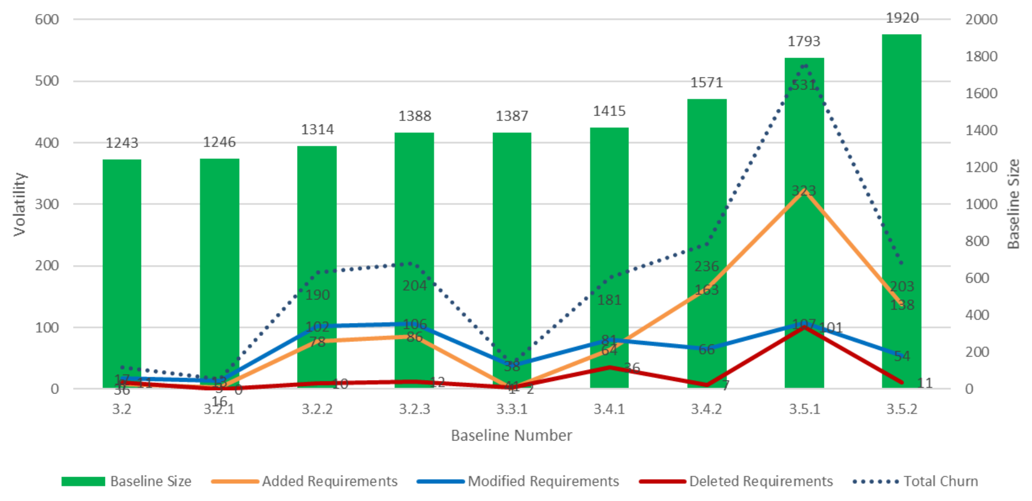


Figure 3.14: Requirements baseline volatility for Product X.

Volatility × **Feature Level** On the feature level, we measure the number of additions, modifications, deletions, and total churn for requirements in a feature in a specific baseline. The number of additions is measured by counting the number of requirements that exist in a given feature (w) in a specific baseline (i) but not in an older baseline (j). Equation 3.36 tests the conditions then Equation 3.37 (M27) counts the number of added requirements per feature resulting in an absolute value. Equation 3.38 calculates the percentage of added requirements in relation to feature size (M28).

$$isAddedFeature(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge FeatureID = w \wedge ReqID \in Baseline_i \wedge ReqID \notin Baseline_j \\ 0, & otherwise \end{cases} \tag{3.36}$$

$$M27 : FeatureAddedA_{wi} = \sum_{k=1}^{n} IsAddedFeature(ReqID_k, w) \tag{3.37}$$

$$M28 : FeatureAddedP_{wi} = \frac{FeatureAddedA_{wi}}{FeatureSize_{wi}} \times 100 \tag{3.38}$$

The number of deletions is measured by counting the number of requirements that exists in a given feature (w) in an older baseline (j) but not in the current baseline (i). Equation 3.39 tests these conditions then Equation 3.40 (M29) counts the number of deleted requirements per feature resulting in an absolute value. Equation 3.41 calculates the percentage of deleted requirements in relation to baseline feature size (M30).

$$isDeletedFeature(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge FeatureID = w \wedge ReqID \notin Baseline_i \wedge ReqID \in Baseline_j \\ 0, & otherwise \end{cases} \tag{3.39}$$

$$M29 : FeatureDeletedA_{wi} = \sum_{k=1}^{n} IsDeletedFeature(ReqID_k, w) \tag{3.40}$$

$$M30 : FeatureDeletedP_{wi} = \frac{FeatureDeletedA_{wi}}{FeatureSize_{wi}} \times 100 \tag{3.41}$$

The number of modifications is measured by counting the number of requirements that are in a given feature (w) in an older baseline (j) and in the current baseline (i), but its text has been modified. Equation 3.42 tests these conditions then Equation 3.43 (M31) counts the number of modified requirements per feature resulting in an absolute value. Equation 3.44 calculates the percentage of deleted requirements in relation to feature size (M32).

$$isModifiedFeature(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge FeatureID = w \\ \wedge ReqID \in Baseline_i \wedge ReqID \in Baseline_j & \\ \wedge ReqText_i \neq ReqText_j & \\ 0, & otherwise \end{cases} \tag{3.42}$$

$$M31 : FeatureModifiedA_{wi} = \sum_{k=1}^{n} IsModifiedFeature(ReqID_k, w) \tag{3.43}$$

$$M32 : FeatureModifiedP_{wi} = \frac{FeatureModifiedA_{wi}}{FeatureSize_{wi}} \times 100 \tag{3.44}$$

Finally, requirements churn on the feature level is measured by adding the number of added, deleted, and modified requirements, which can be achieved using Equation 3.45 (M33) resulting in an absolute value. Equation 3.46 calculates the percentage of requirements churn in relation to feature size (M34).

$$\begin{aligned} M33 : FeatureChurnA_i = \\ FeatureAddedA_i + FeatureDeletedA_i + FeatureModifiedA_i \end{aligned} \tag{3.45}$$

$$M34 : FeatureChurnP_i = \frac{FeatureChurnA_i}{FeatureSize_i} \times 100 \tag{3.46}$$

Where:
w is feature for which volatility is being calculated
i is the baseline that contains the feature for which volatility is being calculated
j is an older baseline number
k is the requirement count in baseline
n is the number of requirements in feature w

Table 3.14 shows sample volatility measures for feature 4.3 for Product X from our data and Figure 3.15 visualizes the data.

Table 3.14: Feature 4.3 volatility for Product X.

Baseline	**M2: Feature_ SizeA**	**M27: Feature_ AddedA**	**M28: Feature_ AddedP**	**M29: Feature_ DeletedA**	**M30: Feature_ DeletedP**	**M31: Feature_ ModifiedA**	**M32: Feature_ ModifiedP**	**M33: Feature_ ChurnA**	**M34: Feature_ ChurnP**
3.2	45	1	2.22%	0	0.00%	1	2.22%	2	4.44%
3.2.1	45	0	0.00%	0	0.00%	0	0.00%	0	0.00%
3.2.2	51	6	11.76%	0	0.00%	2	3.92%	8	15.69%
3.2.3	56	0	0.00%	0	0.00%	0	16.07%	9	16.07%
3.3.1	56	0	0.00%	0	0.00%	1	1.79%	1	1.79%
3.4.1	63	7	11.11%	0	0.00%	7	11.11%	14	22.22%
3.4.2	63	0	0.00%	0	0.00%	0	0.00%	0	0.00%
3.5.1	64	2	3.13%	1	1.56%	19	29.69%	22	34.38%
3.5.2	64	0	0.00%	0	0.00%	0	0.00%	0	0.00%

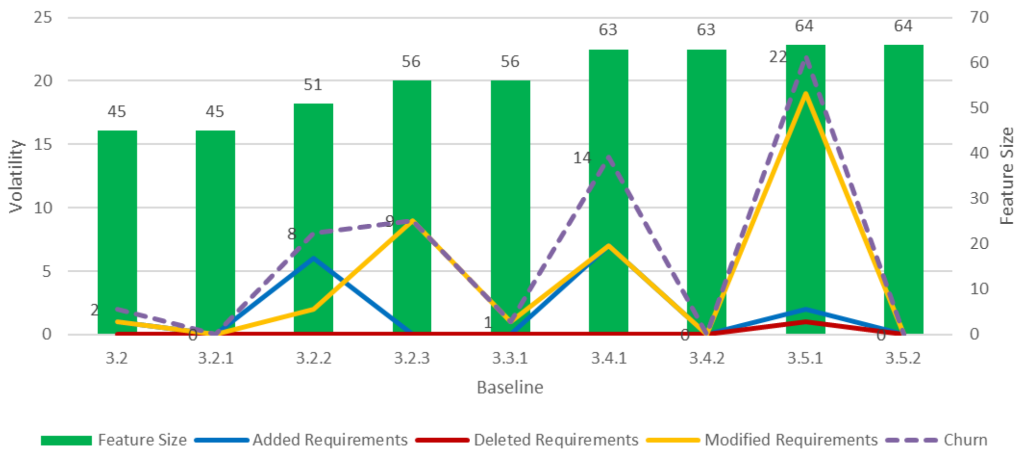


Figure 3.15: Feature 4.3 volatility for Product X.

Volatility × **Release Level** On the release level, we measure the number of additions, modifications, deletions, and total churn for requirements in a release per baseline. The number of additions is measured by counting the number of requirements that exist in a given release (w) in the current baseline (i) but not in an older baseline (j). Equation 3.47 tests the conditions then Equation 3.48 (M35) counts the number of added requirements per release resulting in an absolute value. Equation 3.49 calculates the percentage of added requirements in relation to release size (M36).

$$isAddedRelease(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge Release = w \wedge ReqID \in Baseline_i \wedge ReqID \notin Baseline_j \\ 0, & otherwise \end{cases} \tag{3.47}$$

$$M35 : ReleaseAddedA_{wi} = \sum_{k=1}^{n} IsAddedRelease(ReqID_k, w) \tag{3.48}$$

$$M36 : ReleaseAddedP_{wi} = \frac{ReleaseAddedA_{wi}}{ReleaseSize_{wi}} \times 100 \tag{3.49}$$

The number of deletions is measured by counting the number of requirements that exists in a given release (w) in an older baseline (j) but not in the current baseline (i). Equation 3.50 tests these conditions then Equation 3.51 (M37) counts the number of deleted requirements per release resulting in an absolute value. 3.52 calculates the percentage of deleted requirements in relation to baseline release size (M38).

$$isDeletedRelease(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge Release = w \wedge ReqID \notin Baseline_i \wedge ReqID \in Baseline_j \\ 0, & otherwise \end{cases} \tag{3.50}$$

$$M37 : ReleaseDeletedA_{wi} = \sum_{k=1}^{n} IsDeletedRelease(ReqID_k, w) \tag{3.51}$$

$$M38 : ReleaseDeletedP_{wi} = \frac{ReleaseDeletedA_{wi}}{ReleaseSize_{wi}} \times 100 \tag{3.52}$$

The number of modifications is measured by counting the number of requirements that are in a given release (w) in an older baseline (j) and in the current baseline (i), but its text has been modified. Equation 3.53 tests these conditions then Equation 3.54 (M39) counts the number of modified requirements per release resulting in an absolute value. Equation 3.55 calculates the percentage of deleted requirements in relation to feature size (M40).

$$isModifiedRelease(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge Release = w \\ \wedge ReqID \in Baseline_i \wedge ReqID \in Baseline_j & \\ \wedge ReqText_i \neq ReqText_j & \\ 0, & otherwise \end{cases} \tag{3.53}$$

$$M39 : ReleaseModifiedA_{wi} = \sum_{k=1}^{n} IsModifiedRelease(ReqID_k, w) \tag{3.54}$$

$$M40 : ReleaseModifiedP_{wi} = \frac{ReleaseModifiedA_{wi}}{ReleaseSize_{wi}} \times 100 \tag{3.55}$$

Finally, requirements churn on the release level is measured by adding the number of added, deleted, and modified requirements, which can be achieved using Equation 3.56 (M41) resulting in an absolute value. Equation 3.57 calculates the percentage of requirements churn in relation to release size (M42).

$$M41 : ReleaseChurnA_i = ReleaseAddedA_i + ReleaseDeletedA_i + ReleaseModifiedA_i \tag{3.56}$$

$$M42 : ReleaseChurnP_i = \frac{ReleaseChurnA_i}{ReleaseSize_i} \times 100 \tag{3.57}$$

Where:
w is the release number for which volatility is being calculated
i is the baseline that contains the release for which volatility is being calculated
j is an older baseline
k is the requirement count in baseline
n is the number of requirements in release w

Volatility × Safety Level On the safety level, we measure the number of additions, modifications, deletions, and total churn for requirements in a safety level per baseline. The number of additions is measured by counting the number of requirements that exist in a given safety level (w) in the current baseline (i) but not in an older baseline (j). Equation 3.58 tests the conditions then Equation 3.59 (M43) counts the number of added requirements per safety level resulting in an absolute value. 3.60 calculates the percentage of added requirements in relation to safety level size (M44).

$$isAddedSafety(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge Safety = w \wedge ReqID \in Baseline_i \wedge ReqID \notin Baseline_j \\ 0, & otherwise \end{cases} \tag{3.58}$$

$$M43 : ReleaseAddedA_{wi} = \sum_{k=1}^{n} IsAddedSafety(ReqID_k, w) \tag{3.59}$$

$$M44 : SafetyAddedP_{wi} = \frac{SafetyAddedA_{wi}}{SafetySize_{wi}} \times 100 \tag{3.60}$$

The number of deletions is measured by counting the number of requirements that exists in a given safety level (w) in an older baseline (j) but not in the current baseline (i). Equation 3.61 tests these conditions then Equation 3.62 (M45) counts the number of deleted requirements per safety level resulting in an absolute value. Equation 3.63 calculates the percentage of deleted requirements in relation to baseline safety level size.

$$isDeletedSafety(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge Safety = w \wedge ReqID \notin Baseline_i \wedge ReqID \in Baseline_j \\ 0, & otherwise \end{cases} \tag{3.61}$$

$$M45 : SafetyDeletedA_{wi} = \sum_{k=1}^{n} IsDeletedSafety(ReqID_k, w) \tag{3.62}$$

$$M46 : SafetyDeletedP_{wi} = \frac{SafetyDeletedA_{wi}}{SafetySize_{wi}} \times 100 \tag{3.63}$$

The number of modifications is measured by counting the number of requirements that are in a given safety level (w) in an older baseline (j) and in the current baseline (i), but its text has been modified. Equation 3.64 tests these conditions then Equation 3.65 (M47) counts the number of modified requirements per safety level resulting in an absolute value. Equation 3.66 calculates the percentage of deleted requirements in relation to safety level size (M48).

$$isModifiedSafety(ReqID, w) = \begin{cases} 1, & ReqType = CORE \wedge Safety = w \\ \wedge ReqID \in Baseline_i \wedge ReqID \in Baseline_j & \\ \wedge ReqText_i \neq ReqText_j & \\ 0, & otherwise \end{cases} \tag{3.64}$$

$$M47 : SafetyModifiedA_{wi} = \sum_{k=1}^{n} IsModifiedSafety(ReqID_k, w) \tag{3.65}$$

$$M48 : SafetyModifiedP_{wi} = \frac{SafetyModifiedA_{wi}}{SafetySize_{wi}} \times 100 \tag{3.66}$$

Finally, requirements churn on the safety level is measured by adding the number of added, deleted, and modified requirements, which can be achieved using Equation 3.67 (M49) resulting in an absolute value. Equation 3.68 calculates the percentage of requirements churn in relation to safety level size (M50).

$$\begin{aligned} M49 : SafetyChurnA_i = \\ SafetyAddedA_i + SafetyDeletedA_i + SafetyModifiedA_i \end{aligned} \tag{3.67}$$

$$M50 : SafetyChurnP_i = \frac{SafetyChurnA_i}{SafetySize_i} \times 100 \tag{3.68}$$

Where:

w is the safety level for which volatility is being calculated

i is the baseline that contains the safety level for which volatility is being calculated

j is an older baseline

k is the requirement count in safety level

n is the number of requirements in safety level w

Status Metrics

We define 8 requirements status metrics on the baseline, feature, release, and safety levels. Status metrics use the ReqStatus meta-data item.

Status × Baseline Level To measure baseline status, we count the number of CORE requirements in a given status i. Equation 3.69 tests the conditions then Equation 3.70 (M51) counts the number of requirements per status per baseline resulting in an absolute value. Equation 3.69 calculates the percentage of requirement for a given status in relation to baseline size (M52). Figure 3.16 shows status measures for baseline 4.4 for Product X. We included three statuses for illustrative purposes (more statuses are used): in creation, in review, and approved. We see that 49% of baseline 4.4 is approved and 42% are in review while 9% of baseline requirements are in creation.

$$isStatus(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge ReqStatus = i \\ 0, & otherwise \end{cases} \tag{3.69}$$

$$M51 : BaselineStatusA_{ki} = \sum_{j=1}^{n} IsStatus(ReqID_j, i) \tag{3.70}$$

$$M52 : BaselineStatusP_{ki} = \frac{BaselineStatusA_{ki}}{BaselineSize_{ki}} \times 100 \tag{3.71}$$

Where:

i is a requirements status

k is the baseline number for which status is being calculated

j is the requirement count in baseline

n is the number of requirements in the baseline

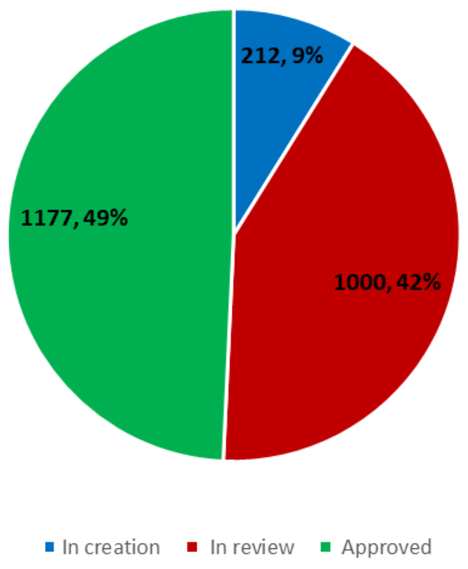


Figure 3.16: Baseline 4.4 status for Product X.

Status × Feature Level To measure feature status, we count the number of CORE requirements in a given status per feature. Equation 3.72 tests the conditions then Equation 3.73 (M53) counts the number of requirements per status per feature resulting in an absolute value. Equation 3.74 calculates the percentage of requirements for a given status in relation to feature size (M54).

$$isStatusandFeature(ReqID, i, w) = \begin{cases} 1, & ReqType = CORE \wedge ReqStatus = i \wedge FeatureID = w \\ 0, & otherwise \end{cases} \tag{3.72}$$

$$M53 : FeatureStatusA_{kiw} = \sum_{j=1}^{n} IsStatusandFeature(ReqID_j, i, w) \tag{3.73}$$

$$M54 : FeatureStatusP_{kiw} = \frac{FeatureStatusA_{kiw}}{FeatureSize_{kiw}} \times 100 \tag{3.74}$$

Where:
i is a requirements status
k is the baseline number
w is the feature ID for which status is being calculated
j is the requirement count in feature
n is the number of requirements in the feature

Status × Release Level To measure release status, we count the number of CORE requirements in a given status per release. Equation 3.75 tests the conditions then Equation 3.76 (M55) counts the number of requirements per status per release resulting in an absolute value. Equation 3.77 calculates the percentage of requirements for a given status in relation to release size.

$$isStatusandRelease(ReqID, i, w) = \begin{cases} 1, & ReqType = CORE \wedge ReqStatus = i \wedge Release = w \\ 0, & otherwise \end{cases} \tag{3.75}$$

$$M55 : ReleaseStatusA_{kiw} = \sum_{j=1}^{n} IsStatusandRelease(ReqID_j, i, w) \tag{3.76}$$

$$M56 : ReleaseStatusP_{kiw} = \frac{ReleaseStatusA_{kiw}}{ReleaseSize_{kiw}} \times 100 \tag{3.77}$$

Where:
i is a requirements status
k is the baseline number
w is the release number for which status is being calculated
j is the requirement count in release
n is the number of requirements in the release

Status × Safety Level To measure safety level status, we count the number of CORE requirements in a given status per safety level. Equation 3.78 tests the conditions then Equation 3.79 (M57) counts the number of requirements per status per safety level resulting in an absolute value. Equation 3.80 calculates the percentage of requirements for a given status in relation to safety level size (M58).

$$isStatusandSafety(ReqID, i, w) = \begin{cases} 1, & ReqType = CORE \wedge ReqStatus = i \wedge Safety = w \\ 0, & otherwise \end{cases} \tag{3.78}$$

$$M57 : SafetyStatusA_{kiw} = \sum_{j=1}^{n} IsStatusandSafety(ReqID_j, i, w) \tag{3.79}$$

$$M58 : SafetyStatusP_{kiw} = \frac{SafetyStatusA_{kiw}}{SafetySize_{kiw}} \times 100 \tag{3.80}$$

Where:
i is a requirements status
k is the baseline number
w is the safety level for which status is being calculated
j is the requirement count in safety level
n is the number of requirements in the safety level

Coverage Metrics

We defined 32 requirements coverage metrics on the baseline, feature, release, and safety levels. Coverage metrics use the *in-links* and *out-links* meta-data items (see Table 3.5). In our project, the *in-links* meta-data item contained *realize* links from the design document to the requirements. The *out-links* meta-data item contained *verifies* links from the requirements to the tests. Below we describe how *realizes* and *verifies* are used in the defined metrics to measure requirements coverage.

Coverage × Baseline Level To measure baseline *design* coverage, we first test whether a requirement is of type CORE and that it contains a *realizes* link in its InLinks meta-data item (Equation 3.81). Equation 3.82 (M59) calculates baseline design coverage by counting the number of requirements that satisfy those conditions for a given baseline resulting in an absolute value. Equation 3.83 (M60) calculates baseline design coverage percentage by dividing the absolute value from Equation 3.82 by baseline size.

To calculate the degree that a baseline is NOT covered by design, we first test whether a requirement is of type CORE and that its InLinks does not contain a 'realizes' link (Equation 3.84). We then count the number of requirements that satisfy those conditions using Equation 3.85 (M61), which results in an absolute value (i.e., the number of requirements in the baseline not covered by design). Equation 3.86 (M62) calculates the percentage of the baseline that is not covered by design.

$$isCoveredDesign(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \in InLinks \\ 0, & otherwise \end{cases} \tag{3.81}$$

$$M59 : BaselineDesignCoverageA_k = \sum_{j=1}^{n} IsCoveredDesign(ReqID_j) \tag{3.82}$$

$$M60 : BaselineDesignCoverageP_k = \frac{BaselineDesignCoverageA_k}{BaselineSize_k} \times 100 \tag{3.83}$$

$$isNotCoveredDesign(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \notin InLinks \\ 0, & otherwise \end{cases} \tag{3.84}$$

$$M61 : BaselineDesignNoCoverageA_k = \sum_{j=1}^{n} IsNotCoveredDesign(ReqID_j) \tag{3.85}$$

$$M62: BaselineDesignNoCoverageP_k = \frac{BaselineDesignNoCoverageA_k}{BaselineSize_k} \times 100 \quad (3.86)$$

Where:
k is the baseline number for which coverage is being calculated
j is the requirement count in baseline
n is the number of requirements in baseline

To measure baseline *test* coverage, we first test whether a requirement is of type CORE and it contains a 'verifies' link in its *OutLinks* meta-data item using Equation 3.87. Equation 3.88 (M63) counts the number of requirements that satisfy those conditions for a given baseline, which results in an absolute value (i.e., the number of requirements in a baseline that are covered by tests). Equation 3.89 (M64) calculates baseline test coverage percentage by dividing the absolute value from Equation 3.88 by baseline size.

Similar to the steps for calculating the degree that a baseline is NOT covered by design, we use Equations 3.90–3.92 to measure the degree a baseline is not covered by tests. M65 results in an absolute value (i.e., the number of requirements in a baseline not covered by test) while M66 results in a percentage.

$$isCoveredTest(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge VERIFIES \in InLinks \\ 0, & otherwise \end{cases} \quad (3.87)$$

$$M63: BaselineTestCoverageA_k = \sum_{j=1}^{n} IsCoveredTest(ReqID_j) \quad (3.88)$$

$$M64: BaselineTestCoverageP_k = \frac{BaselineTestCoverageA_k}{BaselineSize_k} \times 100 \quad (3.89)$$

$$isNotCoveredTest(ReqID) = \begin{cases} 1, & ReqType = CORE \wedge VERIFIES \notin InLinks \\ 0, & otherwise \end{cases} \quad (3.90)$$

$$M65: BaselineTestNoCoverageA_k = \sum_{j=1}^{n} IsNotCoveredTest(ReqID_j) \quad (3.91)$$

$$M66 : BaselineTestNoCoverageP_k = \frac{BaselineTestNoCoverageA_k}{BaselineSize_k} \times 100 \quad (3.92)$$

Where:
k is the baseline number for which coverage is being calculated
j is the requirement count in baseline
n is the number of requirements in baseline

Table 3.15 shows an example of baseline design coverage measures for requirements baseline 4.2.3 for Product X and Table 3.16 shows the test coverage. Figures 3.17 and 3.18 depict the design and test coverage measures for Product X. We see that baseline 4.2.3 has 91% design coverage but only 19% test coverage.

Table 3.15: Baseline 4.2.3 design coverage for Product X.

Baseline	**M1: Baseline_ SizeA**	**M59: BaselineDesign_ CoverageA**	**M60: BaselineDesign_ CoverageP**	**M61: BaselineDesign_ NoCoverageA**	**M62: BaselineDesign_ NoCoverageP**
4.2.3	2090	1903	91.05%	187	8.95%

Table 3.16: Baseline 4.2.3 test coverage for Product X.

Baseline	**M1: Baseline_ SizeA**	**M63: BaselineTest_ CoverageA**	**M64: BaselineTest_ CoverageP**	**M65: BaselineTest_ NoCoverageA**	**M66: BaselineTest_ NoCoverageP**
4.2.3	2090	390	18.66%	1700	81.34%

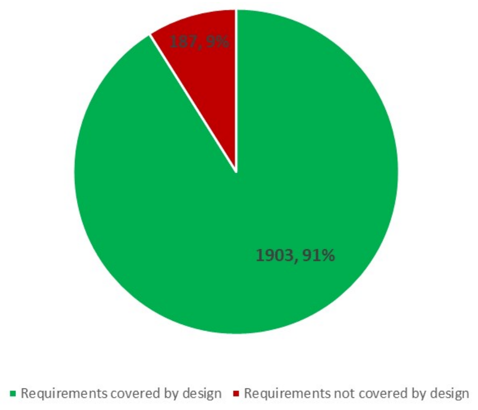


Figure 3.17: Baseline 4.2.3 design coverage.

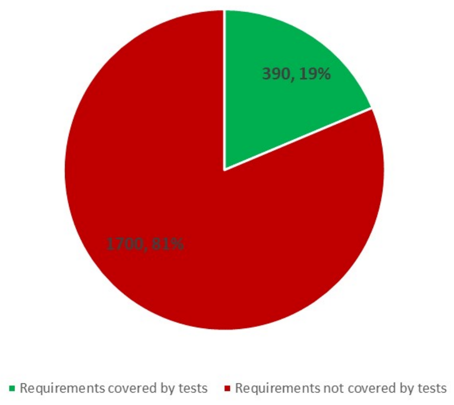


Figure 3.18: Baseline 4.2.3 test coverage.

Coverage × Feature Level To measure the degree a feature is covered by *design*, we first test whether a requirement is of type CORE, that its feature ID matches the feature ID we wish to measure, and that it contains a 'realizes' link in its InLinks (Equation 3.93). Equation 3.94 (M67) calculates feature design coverage by counting the number of requirements that satisfy those conditions for a given feature resulting in an absolute value. Equation 3.95 (M68) calculates feature design coverage percentage by dividing the absolute value from Equation 93 by feature size. Similarly, we use Equations 3.96–3.98 to measure the degree a feature is not covered by design. M69 results in an absolute value while M70 results in a percentage.

$$isCoveredDesignF(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \in InLinks \wedge FeatureID = i \\ 0, & otherwise \end{cases} \tag{3.93}$$

$$M67 : FeatureDesignCoverageA_{ki} = \sum_{j=1}^{n} IsCoveredDesignF(ReqID_j, i) \tag{3.94}$$

$$M68 : FeatureDesignCoverageP_k = \frac{FeatureDesignCoverageA_{ki}}{FeatureSize_{ki}} \times 100 \tag{3.95}$$

$$isNotCoveredDesignF(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \notin InLinks \wedge FeatureID = i \\ 0, & otherwise \end{cases} \tag{3.96}$$

$$M69 : FeatureDesignNoCoverageA_{ki} = \sum_{j=1}^{n} IsNotCoveredDesignF(ReqID_j, i) \tag{3.97}$$

$$M70 : FeatureDesignNoCoverageP_{ki} = \frac{FeatureDesignNoCoverageA_{ki}}{FeatureSize_{ki}} \times 100 \tag{3.98}$$

Where:
k is the baseline number
i is the feature ID for which coverage is being calculated
j is the requirement count in feature
n is the number of requirements in feature

To measure the degree a feature is covered by tests, we follow the same approach we used for feature design coverage above with the only difference being that we are testing whether a requirement's OutLinks contains a 'verifies' link using Equation 3.99. Equations 3.100 (M71) and 3.101 (M72) then calculate the feature test coverage in absolute and relative values. To measure the degree a feature is not covered by test, we use Equations 3.102–3.104. M73 results in an absolute value while M74 results in a percentage.

$$isCoveredTestF(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \in InLinks \wedge FeatureID = i \\ 0, & otherwise \end{cases} \tag{3.99}$$

$$M71 : FeatureTestCoverageA_{ki} = \sum_{j=1}^{n} IsCoveredTestF(ReqID_j, i) \tag{3.100}$$

$$M72 : FeatureTestCoverageP_k = \frac{FeatureTestCoverageA_{ki}}{FeatureSize_{ki}} \times 100 \tag{3.101}$$

$$isNotCoveredTestF(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \notin InLinks \wedge FeatureID = i \\ 0, & otherwise \end{cases} \tag{3.102}$$

$$M73 : FeatureTestNoCoverageA_{ki} = \sum_{j=1}^{n} IsNotCoveredTestF(ReqID_j, i) \tag{3.103}$$

$$M74 : FeatureTestNoCoverageP_{ki} = \frac{FeatureTestNoCoverageA_{ki}}{FeatureSize_{ki}} \times 100 \tag{3.104}$$

Where:

k is the baseline number

i is the feature ID for which coverage is being calculated

j is the requirement count in feature

n is the number of requirements in feature

Table 3.17 shows an example of feature design coverage measures for features 4.18 – 4.23 in Product X (feature names are omitted for privacy reasons). We see that, for example, 9 out of feature 4.18's 9 requirements are covered by design (i.e., 100% feature design coverage) while zero requirements of feature 4.20 have been covered by design (i.e., 0% coverage). Figure 3.19 shows a sample graph that can be generated from the data in Table 3.17. The bar graphs represent the absolute values while the line graphs represent the percentages. Similar measures and graphs can be generated from feature test coverage metrics.

Table 3.17: Feature-design coverage in Product X.

Feature	**M2: Feature_ SizeA**	**M67: FeatureDesign_ CoverageA**	**M68: FeatureDesign_ CoverageP**	**M69: FeatureDesign_ NoCoverageA**	**M70: FeatureDesign_ NoCoverageP**
4.18	9	9	100.00%	0	0.00%
4.19	9	7	77.78%	2	22.22%
4.20	20	9	0.00%	20	100.00%
4.21	32	10	31.25%	22	68.75%
4.22	8	4	50.00%	4	50.00%
4.23	27	10	37.04%	17	62.96%

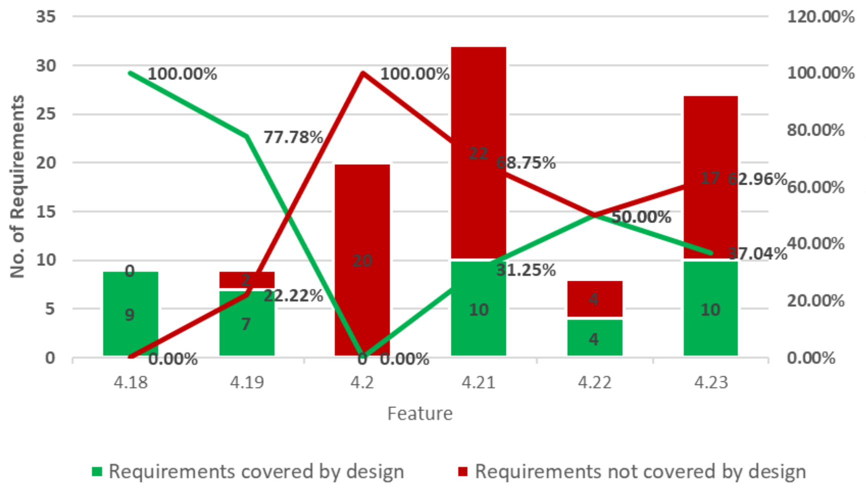


Figure 3.19: Feature-design coverage in Product X.

Coverage × Release Level Similar to the feature level design coverage metrics above, we measure the degree a release is covered by design, by testing whether a requirement is of type CORE, that its release number matches the release we wish to measure, and that it contains a 'realizes' link in its InLinks (Equation 3.105). Equation 3.106 (M75) calculates release design coverage by counting the number of requirements that satisfy those conditions for a given release resulting in an absolute value. Equation 3.107 (M76) calculates feature design coverage percentage by dividing the absolute value from Equation 105 by release size. Similarly, we use Equations 3.108–3.110 to measure the degree a release is not covered by design. M77 results in an absolute value while M78 results in a percentage.

$$isCoveredDesignR(ReqID, i) = \begin{cases} 1, & ReqType = CORE \land REALIZES \in InLinks \land Release = i \\ 0, & otherwise \end{cases} \tag{3.105}$$

$$M75 : ReleaseDesignCoverageA_{ki} = \sum_{j=1}^{n} IsCoveredDesignR(ReqID_j, i) \tag{3.106}$$

$$M76 : ReleaseDesignCoverageP_k = \frac{ReleaseDesignCoverageA_{ki}}{ReleaseSize_{ki}} \times 100 \tag{3.107}$$

$$isNotCoveredDesignR(ReqID, i) = \begin{cases} 1, & ReqType = CORE \land REALIZES \notin InLinks \land Release = i \\ 0, & otherwise \end{cases} \tag{3.108}$$

$$M77 : ReleaseDesignNoCoverageA_{ki} = \sum_{j=1}^{n} IsNotCoveredDesignR(ReqID_j, i) \tag{3.109}$$

$$M78 : ReleaseDesignNoCoverageP_{ki} = \frac{ReleaseDesignNoCoverageA_{ki}}{ReleaseSize_{ki}} \times 100 \tag{3.110}$$

Where:
k is the baseline number
i is the release number for which coverage is being calculated
j is the requirement count in release
n is the number of requirements in release

To measure the degree a release is covered by tests, we follow the same approach we used for release design coverage above with the only difference being that we are testing whether a requirement's OutLinks contains a 'verifies' link using Equation 3.111. Equations 3.112 (M79) and 3.113 (M80) then calculate the release test coverage in absolute and relative values. To measure the degree a release is not covered by test, we use Equations 3.114–3.116. M81 results in an absolute value while M82 results in a percentage.

$$isCoveredTestR(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \in InLinks \wedge Release = i \\ 0, & otherwise \end{cases} \tag{3.111}$$

$$M79: ReleaseTestCoverageA_{ki} = \sum_{j=1}^{n} IsCoveredTestR(ReqID_j, i) \tag{3.112}$$

$$M80: ReleaseTestCoverageP_k = \frac{ReleaseTestCoverageA_{ki}}{ReleaseSize_{ki}} \times 100 \tag{3.113}$$

$$isNotCoveredTestR(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \notin InLinks \wedge Release = i \\ 0, & otherwise \end{cases} \tag{3.114}$$

$$M81: ReleaseTestNoCoverageA_{ki} = \sum_{j=1}^{n} IsNotCoveredTestR(ReqID_j, i) \tag{3.115}$$

$$M82: ReleaseTestNoCoverageP_{ki} = \frac{ReleaseTestNoCoverageA_{ki}}{ReleaseSize_{ki}} \times 100 \tag{3.116}$$

Where:

k is the baseline number

i is the release for which coverage is being calculated

j is the requirement count in release

n is the number of requirements in release

Table 3.18 shows an example of release design coverage measures for releases 1-7 for Product X. We see that release design coverage tells a different story than that by feature design coverage. We have 100% release design coverage for releases 1, 2, 3, 5, and 6 and 94% design coverage for release 4. Release 7 is about 60% percent covered with 170 requirements that are still not covered by design. Figure 3.20 shows a sample graph that can be generated from the data in Table 3.18. Similar measures and graphs can be generated from release test coverage metrics.

Table 3.18: Release-design coverage in Product X.

Release	**M4: Release_ SizeA**	**M75: ReleaseDesign_ CoverageA**	**M76: ReleaseDesign_ CoverageP**	**M77: ReleaseDesign_ NoCoverageA**	**M78: ReleaseDesign_ NoCoverageP**
1	174	174	100.00%	0	0.00%
2	474	474	100.00%	0	0.00%
3	219	219	100.00%	0	0.00%
4	234	220	94.02%	14	5.98%
5	334	334	100.00%	0	0.00%
6	228	228	100.00%	0	0.00%
7	424	254	59.91%	170	40.09%

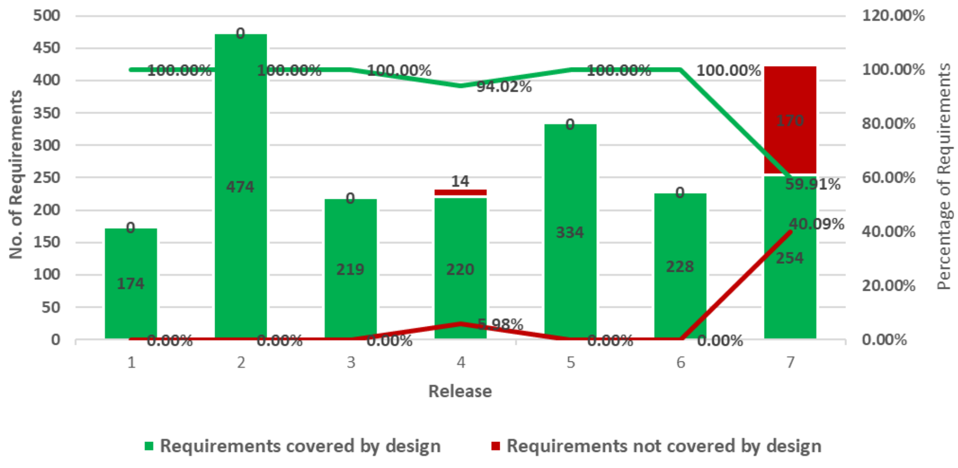


Figure 3.20: Release-design coverage in Product X.

Coverage × Safety Level Finally, on the safety level, we measure the degree a safety level is covered by design by testing whether a requirement is of type CORE, that its safety level matches the safety level we wish to measure, and that it contains a 'realizes' link in its InLinks (Equation 3.117). Equation 3.118 (M83) calculates safety level design coverage by counting the number of requirements that satisfy those conditions for a given safety level resulting in an absolute value. Equation 3.119 (M84) calculates safety level design coverage percentage by dividing the absolute value from Equation 3.118 by safety level size. Similarly, we use Equations 3.120–3.122 to measure the degree a safety level is not covered by design. M85 results in an absolute value while M86 results in a percentage.

$$isCoveredDesignS(ReqID, i) = \begin{cases} 1, & ReqType = CORE \land REALIZES \in InLinks \land Safety = i \\ 0, & otherwise \end{cases} \tag{3.117}$$

$$M83: SafetyDesignCoverageA_{ki} = \sum_{j=1}^{n} IsCoveredDesignS(ReqID_j, i) \tag{3.118}$$

$$M84: SafetyDesignCoverageP_k = \frac{SafetyDesignCoverageA_{ki}}{SafetySize_{ki}} \times 100 \tag{3.119}$$

$$isNotCoveredDesignS(ReqID, i) = \begin{cases} 1, & ReqType = CORE \land REALIZES \notin InLinks \land Safety = i \\ 0, & otherwise \end{cases} \tag{3.120}$$

$$M85: SafetyDesignNoCoverageA_{ki} = \sum_{j=1}^{n} IsNotCoveredDesignS(ReqID_j, i) \tag{3.121}$$

$$M86: SafetyDesignNoCoverageP_{ki} = \frac{SafetyDesignNoCoverageA_{ki}}{SafetySize_{ki}} \times 100 \tag{3.122}$$

Where:
k is the baseline number
i is the safety level for which coverage is being calculated
j is the requirement count in safety level
n is the number of requirements in safety level

To measure the degree a safety level is covered by tests, we follow the same approach we used for safety level design coverage above with the only difference being that we are testing whether a requirement's OutLinks contains a 'verifies' link using Equation 3.123. Equations 3.124 (M87) and 3.125 (M88) then calculate the safety level test coverage in absolute and relative values. To measure the degree a safety level is not covered by test, we use Equations 3.126–3.128. M89 results in an absolute value while M90 results in a percentage.

$$isCoveredTestS(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \in InLinks \wedge Safety = i \\ 0, & otherwise \end{cases} \tag{3.123}$$

$$M87 : SafetyTestCoverageA_{ki} = \sum_{j=1}^{n} IsCoveredTestS(ReqID_j, i) \tag{3.124}$$

$$M88 : SafetyTestCoverageP_k = \frac{SafetyTestCoverageA_{ki}}{SafetySize_{ki}} \times 100 \tag{3.125}$$

$$isNotCoveredTestS(ReqID, i) = \begin{cases} 1, & ReqType = CORE \wedge REALIZES \notin InLinks \wedge Safety = i \\ 0, & otherwise \end{cases} \tag{3.126}$$

$$M89 : SafetyTestNoCoverageA_{ki} = \sum_{j=1}^{n} IsNotCoveredTestS(ReqID_j, i) \tag{3.127}$$

$$M90 : SafetyTestNoCoverageP_{ki} = \frac{SafetyTestNoCoverageA_{ki}}{SafetySize_{ki}} \times 100 \tag{3.128}$$

Where:

k is the baseline number

i is the safety level for which coverage is being calculated

j is the requirement count in safety level

n is the number of requirements in safety level

Table 3.19 shows an example of safety level design coverage measures for regular, safety relevant, and safety critical safety levels in Product X. We see that the safety relevant and safety critical levels are close to 95% covered by design while the regular safety level is close to 90%. Figure 3.21 shows a sample graph that can be generated from the data in Table 3.19. Similar measures and graphs can be generated from safety level test coverage metrics.

Table 3.19: Safety-design coverage in Product X.

Safety Level	**M9: Safety_ SizeA**	**M83: SafetyDesign_ CoverageA**	**M84: SafetyDesign_ CoverageP**	**M85: SafetyDesign_ NoCoverageA**	**M86: SafetyDesign_ NoCoverageP**
Regular Safety	558	500	89.61%	58	10.39%
Safety Relevant	115	109	94.78%	6	5.22%
Safety Critical	487	460	94.46%	27	5.54%

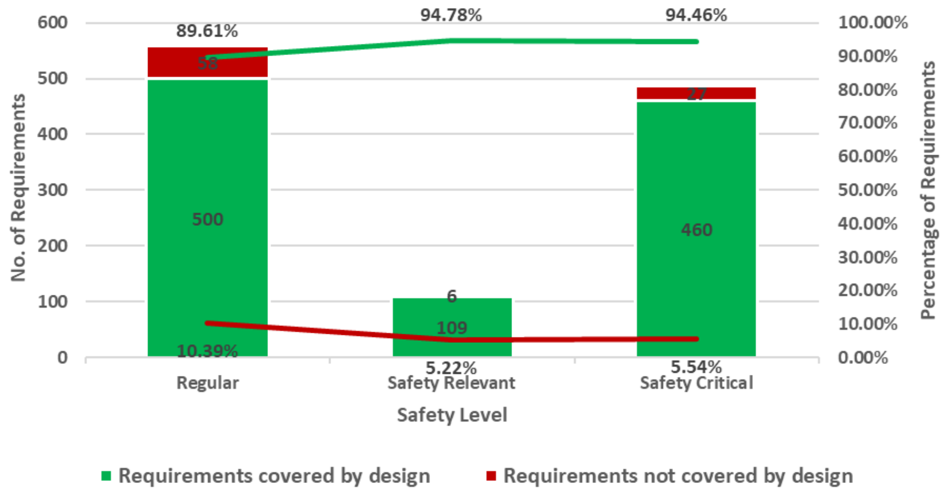


Figure 3.21: Safety-design coverage in Product X.

3.5 Validation of Structured Metrics Suite

As discussed in the Research Method Section (Section 3.3.2), the nature of an AR study entails the subsumption of the empirical validation of the AR findings within the AR study. However, due to the variety of the study's results (metrics, meta-data, attributes, metric levels), different validation sub-processes were carried out for the results of each research question. Particularly, through expert opinion [Yousuf, 2007; Helmer, 1967], content analysis, and observing the use of the findings within the organization's process. Table 3.20 shows the empirical and theoretical validation processes for the findings of each research question.

Table 3.20: Overview of validation of AR study results.

Research Question	**Result**	**Empirical Validation**	**Theoretical Validation**
RQ1	Requirements metrics	AR: usage scenarios of measure by internal stakeholders	Kitchenham's theoretical validation framework [Kitchenham, 1995]
RQ2	Requirements meta-data	AR: content analysis	Comparative analysis with established literature
RQ3	Requirements attributes	AR: expert opinion	Comparative analysis with established literature
RQ4	Requirements metric levels	AR: expert opinion, integration into RM process and documentation	—

3.5.1 Metrics Validation (RQ1)

Validation of software metrics consist of theoretical and empirical validation [Briand et al., 1995]. Theoretical validation is "concerned with demonstrating that a measure is measuring the concept it is purporting to measure" [Briand et al., 1995]. In other words, measure of the claimed attribute is an appropriate numerical characterization by demonstrating that the measure does not abuse any essential properties of the measurement elements [Srinivasan and Devi, 2014]. On the other hand, empirical validity is concerned with a metric's usefulness in practice, which is usually gauged by relating it to an external attribute (e.g., quality). In other words, for a metric to be empirically valid, it must have some predictive power of an external attribute [Antinyan et al., 2016]. For example, a complexity measure is always defined in relation to some external attribute such as error-proneness and effort [Briand et al., 1995]. We describe the theoretical and empirical validation processes of our metrics in the following two sections, respectively.

Theoretical Validation of Metrics

Several software metric validation frameworks have been proposed [Schneidewind, 1992; Briand et al., 1995; Kitchenham, 1995; Srinivasan and Devi, 2014] that focus on code metrics such as coupling, cohesion, and complexity. We use Kitchenham's validation framework [Kitchenham, 1995] due to it general applicability to attributes of different entities. First, elements of the measurement process and their properties that underpin the notion of measurement validity must be defined. Table 3.21 shows the main measurement elements, their descriptions based on Kitchenham's definition of validity [Kitchenham, 1995] and the element's values within the context of our defined metrics.

Table 3.21: Main measurement elements, descriptions, and values according to [Kitchenham, 1995].

Measurement Element	**Description**	**Element Values**
Entity	Objects we observe in the real world. One of the goals of measurement is to capture their characteristics and manipulate them in a formal way. Software entities may be products, processes, or resources of different types.	Requirements set
Attributes	Attributes are the properties that an entity possesses. An entity may possess many attributes, while an attribute can qualify many different entities.	Size, growth, volatility, status, coverage
Units	A measurement unit determines how we measure an attribute. An attribute may be measured in one or more units and the same unit may be used to measure more than one attribute.	Requirements
Scale Types	The most common scale types are: nominal, ordinal, interval, and ratio.	Ratio

Properties of Values	Valid measure are to be defined over a set of permissible values. A set of permissible values may be finite or infinite, bounded or unbounded, discrete or continuous.	Discrete, non-negative integers;
		Continuous, non-negative real numbers;
		Discrete, positive and negative integers;
		Continuous, positive and negative real numbers

Once the main measurement elements have been defined, according to Kitchenham [Kitchenham, 1995], the following criteria must be satisfied in order to decide whether a metric is valid:

1. "For an attribute to be measurable, it must allow different entities to be distinguished from one another." In other words, when measuring an attribute for two different entities that possess said attribute in different degrees, then a valid metric must result in different measures for the two entities.

2. "A valid measure must obey the Representation Condition. Meaning, it must preserve our intuitive notions about the attribute and the way in which it distinguishes different entities."

3. "Each unit of an attribute contributing to a valid measure is equivalent."

4. "Different entities can have the same attribute value (within the limits of measurement error)."

To avoid redundancy, we demonstrate the theoretical validity of two metrics (M2, M68) only. However, the process applies to all 90 metrics.

M2 ($FeatureSizeA_i = \sum_{j=1}^{n} IsCOREandFeature(ReqID_j, i)$) conforms to Kitchenham's properties as follows:

Suppose that we have baseline X. Assume that we have two features in baseline X: F1 and F2. Suppose that F1 has E1 entities (set of requirements), which are $r1, r2, ..., r_m$ and F2 has E2 entities (set of requirements), which are $r1, r2, ..., r_n$.

Property 1: When $m \neq n$ then $FeatureSizeA_{F1} \neq FeatureSizeA_{F2}$. Thus, M2 allows us to distinguish between the attribute (i.e., size) of two entities (i.e., set of requirements in F1 and set of requirements in F2).

Property 2: When $n > M$ then $FeatureSizeA_{F2} > FeatureSizeA_{F1}$. Thus, M2 obeys the representation condition by preserving our intuitive notions about the attribute (i.e., size). In other words, M2 preserves our intuitive notion that if the size of feature F2 is bigger than the size of F1 then the results of M2 for F2 will be larger than the results of M2 for F1.

Property 3: When $m = n$ and $m+1 = n+1$ then $FeatureSizeA_{F1} = FeatureSizeA_{F2}$. Thus, M2 ensures that each unit (i.e., requirement) of an attribute (i.e., size) contributing to a valid measure is equivalent. In other words, if F1 and F2 are equal in size and we added one requirement to F1 and one requirement to F2, then F1 and F2 will remain equivalent because our unit (i.e., requirement) contributed equally to our measure of feature size.

Property 4: When $m = n$ then $FeatureSizeA_{F1} = FeatureSizeA_{F2}$. Thus, M2 allows different entities (i.e., set of requirements in F1 and set of requirements in F2) to have the same attribute (i.e., size) value. Therefore, M2 is theoretically valid according to Kitchenham's metric validation framework.

Similarly, M68 ($FeatureDesignCoverageP_k = \frac{FeatureDesignCoverageA_{ki}}{FeatureSize_{ki}} \times 100$) conforms to Kitchenham's properties as follows:

Suppose that we have baseline X and that baseline X has E entities (i.e., set of requirements), which consist of $r1, r2, ..., r_i$. Assume that we have two features in baseline X: F1 and F2. Suppose that F1 has E1 entities (set of requirements) that satisfy the condition $ReqType = CORE \wedge REALIZES \in InLinks \wedge FeatureID = i$, which are $r1, r2, ..., r_m$ and F2 has E2 entities (set of requirements) that satisfy the condition $ReqType = CORE \wedge REALIZES \in InLinks \wedge FeatureID = i$, which are $r1, r2, ..., r_n$.

Property 1: When $m \neq n$ then
$FeatureDesignCoverageP_{F1} \neq FeatureDesignCoverageP_{F2}$. Thus, M68 allows us to distinguish between the attribute (i.e., coverage) of two entities (i.e., set of requirements in F1 and set of requirements in F2).

Property 2: When $n > m$ then
$FeatureDesignCoverageP_{F2} > FeatureDesignCoverageP_{F1}$. Thus, M68 obeys the representation condition by preserving our intuitive notions about the attribute (i.e., coverage). In other words, M68 preserves our intuitive notion that if the coverage of feature F2 is bigger than the coverage of F1 then the results of M68 for F2 will be larger than the results of M68 for F1.

Property 3: When $m = n$ and $m+1 = n+1$ then
$FeatureDesignCoverageP_{F1} = FeatureDesignCoverageP_{F2}$. Thus, M68 ensures that each unit (i.e., requirement) of an attribute (i.e., coverage) contributing to a valid measure is equivalent. In other words, if F1 and F2 are equal in coverage and we added one requirement

that satisfies the above conditions to F1 and one requirement that also satisfies the above condition to F2, then F1 and F2 will remain equivalent because our unit (i.e., requirement) contributed equally to our measure of feature coverage.

Property 4: When $m = n$ then $FeatureDesignCoverageP_{F1} = FeatureDesignCoverageP_{F2}$. Thus, M68 allows different entities (i.e., set of requirements in F1 and set of requirements in F2) to have the same attribute (i.e., coverage) value. Therefore, M68 is theoretically valid according to Kitchenham's metric validation framework.

Empirical Validation of Metrics

Empirical validation answers the following question: "Is the measure useful in a particular development environment, considering a given purpose in defining the measure?" [Briand et al., 1995]. To answer this question, the relation between an internal attribute measure (e.g., size) and an external attribute measure (e.g., quality) has usually been studied using statistical analysis and modeling techniques [Briand et al., 1995] that are chosen depending on the scale types. However, [Antinyan et al., 2016] argue that when it is not possible to acquire accurate measures for external attributes and, in turn, the validation of internal attributes cannot be conducted by the help of statistical models, then action research can be successfully applied for empirical validation of metrics. Because action research relies on the define-refine-redefine process with practitioners, it shapes the final definition of the measure, which is either accepted or rejected for further application. This process relies on the metric designer – reference group infrastructure. The metric designer is the person assigned to define and validate the metrics (e.g., researcher, data analyst). The reference group is a group of practitioners who are working closely with the measured artifacts (e.g., requirements engineers, architects, testers) and who provide qualitative feedback on the metrics. In our case, the metric designer was the researcher (principal author) and the internal stakeholders were the reference group headed by the main requirements engineering person in the company. Several recent studies have used AR studies to empirically validate metrics (e.g., [Antinyan and Staron, 2017] and [Carvalho et al., 2018]).

In this subsection, we demonstrate the metrics' empirical validity by narrating example usage scenarios from the project to assess their usefulness within our context given the objective of the defined metrics: *Enable the tracking, monitoring, and management of requirements and requirements related-information so as to provide internal stakeholders with requirements-related information that would address their concerns and aid them in their respective process activities.* As discussed in Section 3.3, we identified the need for requirements metrics and subsequently selected, defined, applied, and enhanced the metrics with the internal stakeholders throughout the AR study. The metrics were initially presented in two focus groups within the first two months then updated and generated every month and disseminated through email and upon request to the internal stakeholders of three products (see Table 3.2). Thus, the metrics designer was able to acquire usage scenarios from the internal stakeholders through discussions with the internal stakeholders about their concerns, emails exchanges, and observe the use of metrics within the development process. Moreover, the metrics designer conducted an interview with the systems manager to assess the usability of the metrics.

Table 3.22 consists of example usage scenarios that show how specific metrics were used by which internal stakeholder in the project. The *Metric* column contains the metric ID and title per the metrics defined in Section 3.4.4. In the *Usage Scenarios of Measure from Project* column, each usage scenario is identified with an *S* and the internal stakeholders involved in the scenario are *italicized*.

We note that due to the large size of the projects, the numerous internal stakeholders, large number of metrics, and the uncontrolled nature of an AR study, it was not possible make an inventory of the uses of all metrics. Thus, it is likely that there are other usage scenarios that we did not capture. We then discuss some general observations of the metrics' usage within the project.

Table 3.22: Example usage scenarios of metrics from projects.

Metric	**Usage Scenarios of Measure from Project**
M1: Baseline Size (absolute)	S1- M1 constituted the basis for all other measures. The *systems manager* started presentations with this measure in meetings to provide an estimate of the overall product size for all involved internal stakeholders. S2- The *product manager* used it for resource management and project scheduling; larger product sizes require more staff, time, and effort.
M2: Feature Size (absolute)	S3- *Developer* used feature size to guide his/her development-related tasks such as deciding how many hours will be needed to develop feature X or how many requirements/day must be developed to deliver feature X by the release deadline.
M4: Release Size (absolute)	S4- *Developers* and *designers* used M4 to gauge the amount of time and effort required to design and develop the requirements within a given release before it is scheduled for delivery to the customer. S5- *Systems and product managers* used M4 for release planning (i.e., allocating requirements to specific product releases).
M5: Release Size (percentage)	S6- *Systems and product managers* used it to communicate with the client and higher management (e.g., program manager) and to make strategic decisions about project schedule. In other words, it allowed them to communicate to the customer and higher management the percentage of the product to be delivered for each product release.

M9: Safety Level Size (absolute)	S7- Given the safety criticality of the products, the *safety manager* used it for managing safety requirements in general to answer questions such as: how many requirements of product X are safety critical requirements?
M13: Feature Growth (absolute)	S8- M13 was shown in the second focus group to identify the features with the most growth so *developers* in charge of those features are aware of the new requirements and it was incorporated in the monthly reports for the same purpose.
M26: Baseline Churn (percentage)	In the interview with the *systems manager*, he indicated that volatility measures in general are by far the most important measure for requirements management as they indicate whether the project is stable in relation to its stage. High volatility rates, represented by M26, for example, at later stages of the project indicate problems that need to be resolved. Thus, M26 was used by the *systems manager* to gain a general, big picture view of the state of requirement changes.
M23: Modified req. per Baseline (absolute)	S10- While M25 and M26 were used to give a big picture of requirements changes in a project, metrics like M23 were a catalyst for the *systems and requirements engineer* to investigate deeper into the nature of the change. In one instance, one baseline had an unnaturally large number of modified requirements, but upon further investigation, it was found that the baseline was created before a major product release. Thus, the majority of modifications were due to spelling and grammatical changes to the requirements text as opposed to change in functionality.
M19: Added req. per Feature (absolute) **M20:** Deleted req. per Feature (absolute)	S11- In one metric report, M19 indicated that there was a large number of added requirements to one pf the product's features. However, upon looking at M20, it was made clear that the large number of requirements of added requirements were the same number deleted from another feature. Meaning, the requirements were simply moved from one feature to another. Such insight would not have been possible to gain had we been using M26 only (churn on the baseline level).
M31: Modified Req. per Feature (absolute)	S12- *Developers* used this measure to ensure that modifications in the feature requirements have also been changed in the code.
M51: Baseline Status (absolute) **M52:** Baseline Status (percentage)	S13- While these measures has been requested by the *systems manager* (i.e., reference group) and agreed upon with the metric designer from the start of the AR study, it was not highly useful in practice because the status of such a large number of requirements was not kept up to date on a regular basis. Thus, the measures did not reflect reality. An automatic method must be implemented to update the requirements status to make use of this metric.
M59: Baseline Req.-Design Coverage (absolute) **M61:** Baseline Req.- Design No Coverage (absolute)	S14- The *architect* used M59 and M61 to ensure that design and architect tasks are aligned with the requirements. The objective was to increase requirement-design coverage. S15- The *systems and product managers* used this measure to track progress of the project in terms of schedule and deliverables. A low requirements-design coverage closer to a release deadline is symptomatic of an underlying problem that requires some action.

M67: Feature Req. - Design Coverage (absolute) **M69:** Feature Req. - Design No Coverage (absolute)	S16- These measures were specifically requested by the *architect* who used them to monitor requirements-design on a more granular level. More specifically, if there is a specific feature that is due to be delivered within a given deadline, the architect used this measure to ensure requirement-design coverage for that feature is 100 by the deadline.
M60: Baseline Req. - Design Coverage (percentage) **M68:** Feature Req. - Design Coverage (percentage) **M76:** Release Req. - Design Coverage (percentage)	S17- The *architect* requested these measures usually before a meeting with high-level managers such as the regional R&D manager to demonstrate the progress of the project. S18- The *systems manager* presented these measures to the internal stakeholders of a project in meetings to provide them with a bigger picture of the project progress.
M63: Baseline Req. – Test Coverage (absolute) **M65:** Baseline Req. – Test No Coverage (absolute)	S19- The *testers and test managers* used it to track requirements-test coverage and manage testing tasks accordingly. Lower requirements-test coverage closer to a delivery date was a call to action. S20- The *quality manager* used it for quality management purposes.
M79: Release Req. – Test Coverage (absolute) **M81:** Release Req. – Test No Coverage (absolute)	S21- For each product release, the *systems manager* used these measures to communicate with the clients the state of requirements-test case coverage, which was also used by the tester on the client side. The measures were usually presented with the list of requirements and test cases for that release and whether the tests passed or failed.

General Observations on Metric Usage

First, the sample of usage scenarios in Table 3.22 demonstrated that the objective of metrics we stated above was addressed. In other words, the measures enabled internal stakeholders to track, monitor, and manage requirements and requirements related-information so as to provide them with the requirements-related information that addressed their concerns and aided them in their respective process activities (i.e., architecting, testing, release planning, safety management, quality management).

Second, we can see that the usage of the absolute and relative measures differed. Absolute measures were mostly used by technical and mid-level managers (e.g., architects, developers, product managers) to aid their tasks (e.g., S3, S4, S14, S16) while relative measures were used to communicate to higher-level managers (e.g., regional R&D manager, program manager) and clients to provide bigger pictures of progress, status, and possible problems (e.g., S6, S17). We captured these interactions in a meta-model that we conducted in a separate post analysis of the AR study (Chapter 6 and published in [Noorwali et al., 2019]).

Third, some scenarios indicate a need for supplementing some of the metrics with further, descriptive metrics. For example, M1 (baseline size) can be supplemented with other measures that would provide in-depth information about the quality and complexity of individual requirements [Génova et al., 2013; Antinyan and Staron, 2017]. While M1 provides a number of requirements. Similarly, S10 shows a need to supplement M23 (the number of modified requirements per baseline) with a means to analyze the type of change that occurred to the requirements as not all modifications are equal; a textual change does not require action like a functionality change.

Fourth, we observed that while the reports were generated monthly by the metrics designer, there were requests for requirement metric reports in between the monthly cycles. Thus, the importance of automating them so that the internal stakeholders would have access to the requirements on demand became evident (automation of the metrics is discussed in Chapter 7).

3.5.2 Meta-Data Validation (RQ2)

While a subset of the meta-data items were already in use (e.g., ReqID, ReqText, FeatureID) in all of the projects, others were added to enable the application of the metrics (e.g., release number). To acquire empirical evidence of the meta-data items' use in practice, we first conducted a content analysis of the three project's requirements repositories and ensured that the meta-data items were consistent across all three projects. Thus, the identified meta-data items were applied to the three projects in Table 3.2. Moreover, we examined a fourth project that is within the organization but not within the overall program we were involved in and examined the applicability of the meta-data items to its requirements repository. We found that the repository consisted of ReqID, ReqText, FeatureID, ReqStatus, ReqType, and Safety. Thus, we can deduce that the meta-data items are applicable to projects outside the projects under study.

In addition to the applicability of the meta-data items to various projects, we must also assess whether they enabled the application of the metrics from RQ1 (see Section 3.3.2 and Figure 3.2 for research questions and research question hierarchy). We can see that the metrics reported in Section 3.4.4 utilized the meta-data items identified in Section 3.4.3. The application of metrics within the projects resulted in numerous requirements measures reports. Table 3.23 shows the number of requirements measures reports generated, thus far, for each of the three projects in Table 3.2 and a fourth project that was added to the overall program after the AR study.

Table 3.23: Number of generated requirements measurement reports for projects.

Project	**Number of Measurement Reports**
P1	5
P2	7
P3	20
P4	3

Theoretically, we compared the meta-data items we identified with those found in the literature. We found support for ReqID [Kotonya and Sommerville, 1998; Wiegers, 2006; IEEE, 2011; Ebert and Dumke, 2007], ReqText [Kotonya and Sommerville, 1998; Wiegers, 2006; IEEE, 2011; Ebert and Dumke, 2007], ReqStatus [Kotonya and Sommerville, 1998], and release [Wiegers, 2006].

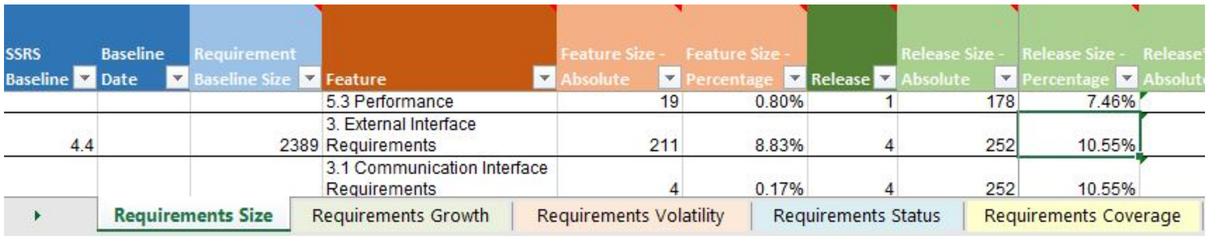
3.5.3 Attributes Validation (RQ3)

Theoretical validation of the identified attributes was the main validation process for the requirements attributes given their theoretical nature. The five requirements attributes we identified are rooted in the literature. While we included a discussion of each of the attributes in Section 3.4.1, we summarize the theoretical underpinnings of the identified attributes in Table 3.24.

Table 3.24: Theoretical underpinnings of identified requirements attributes.

Requirement Attribute	**Theoretical Support**
Size	[Goti, 1998]; [Jones, 2000]; [Loconsole, 2001]; [Loconsole, 2003];[Wiegers, 2006]
Growth	[Boehm, 1991]; [Jones, 1996b,a]; [Kulk and Verhoef, 2008]; [Park et al., 2010]
Volatility	[Nurmuliani et al., 2004]; [Costello and Liu, 1995]; [Thakurta and Ahlemann, 2010]; [Wiegers, 2006]; [Ferreira, 2002; Ferreira et al., 2009, 2011]
Status	[Wiegers, 2006]; [Ebert and Dumke, 2007]
Coverage	[Gotel and Finkelstein, 1994]; [Lormans and van Deursen, 2005; Lormans and Van Deursen, 2006]; [Lago et al., 2009]

In addition to the theoretical validation, we sought expert opinion [Helmer, 1967; Yousuf, 2007] from the internal stakeholders within the projects to 1) ensure that our understanding of the requirements attributes is consistent with the internal stakeholders' understanding and 2) establish the use of the attributes within the RM process in the organization. The main requirements person within the projects reviewed and approved the use of the requirements attributes. In addition, upon identification of the attributes, the requirements measures reports were structured according to the attributes as depicted in Figure 3.22.



SSRS Baseline	Baseline Date	Requirement Baseline Size	Feature	Feature Size - Absolute	Feature Size - Percentage	Release	Release Size - Absolute	Release Size - Percentage	Release Absolut
			5.3 Performance	19	0.80%	1	178	7.46%	
4.4		2389	3. External Interface Requirements	211	8.83%	4	252	10.55%	
			3.1 Communication Interface Requirements	4	0.17%	4	252	10.55%	

Figure 3.22: Screenshot of a requirements measures report.

3.5.4 Metric Levels Validation (RQ4)

Given the novelty of the concept of requirement metric levels, we could not validate our identified levels with the established literature. However, we validated our findings through the internal stakeholders' expert opinions [Helmer, 1967; Yousuf, 2007]. Similar to the requirements attributes, the requirement metric levels were validated by a systems manager, product R&D manager, and quality manager. Moreover, the levels have been incorporated into the requirements measurement process by using them to structure the measures reports as seen in Figure 3.22 and including their associated metrics' definitions in the internal requirements management plan.

3.5.5 Threats to Validity

We discuss the study's validity threats and how we mitigated them according to Runeson and Host's guidelines [Runeson and Höst, 2009].

Construct Validity

Construct validity concerns the operationalized constructs of the study in that whether or not they accurately represent the real-world phenomena. It is possible that the study constructs (i.e., requirements attributes, levels, meta-data, and metrics) might not have been captured accurately by the researcher. The two constructs that are subject to the least construct validity threat are requirements attributes and meta-data. The requirements attributes derived from the AR study were validated against the scientific literature (see Section 3.4.1) and we found support for all the identified attributes in the literature. The second construct that is unlikely to have been inaccurately captured by the researcher is the identified meta-data items as they were selected as-is from the project data. However, the requirement levels and metrics are subject to a higher degree of construct validity threats. There is a possibility that the metrics did not capture what we really intended to measure. At the beginning of the metric definition process, we consulted the literature for available requirements metrics (see Table 3.1 and Section 3.2.3). While the literature on requirements metrics is scant, as shown in Section 3.2.3, the idea of the metrics were rooted in the scientific literature. Moreover, we sought continuous feedback from the internal stakeholders of the project during and after the metric definition process, which we used to continuously enhance the metrics until they were accepted. Finally, the requirements levels we identified are possibly the most subjective findings due to their novelty and a lack of literature on this topic. Although we validated the identified levels with our industrial partner (Section 3.5.4) and demonstrated their use in Section 3.5, we encourage further validation by others.

Internal Validity

Internal validity is concerned with the validity of causal relationships, typically in scientific experiments. Given that our study objective does not include investigation of causal relationships, this threat is not relevant to our study.

External Validity

External validity is concerned with the generalizability of the results to other contexts. We believe the biggest issue of this study concerns the generalizability of its results. The AR study was conducted within the safety-critical, transportation domain and the data were gathered from large-scale projects that consisted of thousands of requirements and employed extensive documentation. Moreover, the requirements engineering and management processes are highly structured and mature in comparison to smaller companies where, for example, agile processes are adopted. These factors affect the generalizability of the results in several ways. First, smaller projects that do not have the same number of requirements may not benefit from the metrics. Second, even larger projects that have large number of requirements may not have the requirements tools nor the documentation in our project, thus it would be hard to define metrics and identify meta-data. Thus, readers must interpret and reuse the results in other contexts with caution. Despite this limitation, the results constitute an important data-point for making scientific progress. Further validation of the findings in different domains and project sizes is encouraged in order to improve their generalizability. However, we believe that requirements are ubiquitous regardless of their form and the essence of our results can still be adapted to different contexts. For example, smaller projects may need the metrics on the baseline level only. Moreover, while the attributes levels, and metrics identified in this study are limited to our context, the idea of identifying these items is universal. Thus, other projects in different contexts would need to measure different requirements attributes at different levels and use different meta-data. It would be interesting to compare between the values of the different domains.

Reliability

Reliability is concerned with the degree of repeatability of the study, which constitutes another issue in our study. While the AR study followed AR principles for software engineering [Susman and Evered, 1978; Santos and Travassos, 2011] to ensure rigor during the study, a level of subjectivity is inevitable during an AR study. The researcher dealt with a plethora of data, internal stakeholders, and was immersed in the specific context of the study, which may make it impossible to achieve perfect repeatability. To deal with this threat, we kept record of project data, project personnel, emails, interviews, meetings and observations. We also retained historical data of the defined metrics from the first version of the metric report to the current version, which shows the evolvement of the results.

Moreover, the analysis techniques of the data were highly dependent on the researcher who had to sift through large amounts of data and identify requirements attributes, levels, metrics, and meta-data items. While the process of identifying the metrics and meta-data enjoys a higher level of repeatability due to their objectivity and dependency on tangible data (i.e., requirements, meta-data items), the process of arriving at the requirements attributes and levels was less objective. However, our use of extensive documentation helped mitigate this threat as shown in Section 3.3. We showed how the application of a metrics validation framework led to the identification of the requirements attributes. The organization and reorganization of the metrics into similar clusters led to the identification of the requirements levels. Nevertheless, we concede that another researcher may possibly arrive at different requirements levels.

3.6 Implications

In this section, we discuss the implications of our findings on practice and research.

3.6.1 Implications for Practice

RE Documentation Practices. Data collection is a prerequisite for measurement [Costello and Liu, 1995; Ebert and Dumke, 2007; Fenton and Biemen, 2015; Kerzner, 2017] and the quality of any measurement process is dependent on careful data collection. For example, *percentage of requirements that have been modified per feature per baseline* is a metric that would require a requirement ID, feature ID, release number and requirement text. Thus, since the early days of SE measurement, it has been recognized that “data should be collected with a clear purpose in mind. Not only a clear purpose but also a clear idea as to the precise way in which they will be analyzed so as to yield the desired information.” [Moroney, 1968]. Costello and Liu [Costello and Liu, 1995] argue that data gathering must be part of the metric definition process and without it the metrics are useless. However, the overhead associated with gathering the data required for measurement is large and has been recognized as a significant barrier to adopting software metrics [Kerzner, 2017]. The problem is further exacerbated in large-scale systems where data is large and distributed across numerous projects and artifacts [Fenton and Biemen, 2015]. Thus, a first step towards facilitating the data gathering procedure for measurement is to know what data to gather. Thus, our identification of the requirements meta-data (see Section 3.4.3) that are required to apply the presented metrics ensures that projects interested in applying the metrics maintain, at the minimum, the identified meta-data items for their requirements.

Requirements Management and Measurement Processes. When discussing good requirements management practices, requirements measurement is usually recommended but treated briefly [Wiegers, 2006; Chemuturi, 2013] without the detail and attention given to other requirements management activities such as change management [Leffingwell and Widrig, 2000; Jayatilleke and Lai, 2018] and traceability [Gotel and Finkelstein, 1994; Gotel et al., 2012; Cleland-Huang et al., 2012]. The lack of detail hinders the integration of requirements measurement into the requirements management process. Thus, the comprehensive requirements metrics suite presented in this paper provides practitioners with the groundwork and details necessary to integrate requirements measurement into the requirements management process.

Customization and Adaptability. One of the success factors of measurement programs is its flexibility and adaptability to the specific context in which it is being implemented [Offen and Jeffery, 1997]. Our delineation of the four elements of requirements measurement allows for the customization of our results for different contexts. An organization need not measure the five attributes we identified using our metrics at the four metric levels; they can be used as guidelines to identify other attributes and define suitable metrics that meet the organization's needs. For example, a project may use the identified attributes as guidelines to define their own metrics or introduce other metric levels (e.g., individual requirement level) that better address their information needs. Thus, the four elements of measurement we identified can be used as a requirements measurement 'schema' that guides the process of requirements measurement while allowing the project or organization to define its details.

Stakeholder Information Needs. It is often difficult to define metrics that satisfy the needs of all interested stakeholders (e.g., general management, project management, developers, requirements engineers, architects, etc.). Thus, defining varying metric levels can aid in addressing the different information needs for different stakeholders [Fenton and Biemen, 2015]. Especially in the context of a large systems-project that consists of numerous stakeholders and processes, our experience indicates that the levels and attribute-level combinations can yield metrics that address different internal stakeholder needs. Projects can experiment within their specific contexts with the different attribute-level combinations to identify which combinations are relevant to which stakeholder group.

Measurement Breadth and Coverage. Metric levels in general ensure breadth and comprehensive coverage of what is to be measured [Costello and Liu, 1995]. For example, if we are interested in defining metrics for all aspects of a systems engineering project, then we need to identify the levels of the systems engineering project that require different metrics such as product and process metrics or system-level metrics and software-level metrics. These levels can be further broken down into sub-levels to ensure further coverage (e.g., product metrics can be categorized into requirements, design, code, and test metrics). Thus, the metric levels presented in this paper can aid practitioners in improving the breadth of requirements measurement in their organizations. And in reference to the customization and adaptability discussion above, further metric levels can be identified to ensure measurement breadth and coverage.

RE Dashboards and Measurement Automation. In general, measurement should be fully automated to fully exploit the potential of metrics, which is usually achieved through dashboards [Pauwels et al., 2009; Selby, 2009]. Dashboards reduce the overhead associated with collecting data, applying the metrics, generating graphs and visualizations, and keeping the measures up-to-date [Selby, 2007, 2009]. Moreover, dashboards allow wider sharing and communication of data across an organization [Pauwels et al., 2009]. This becomes especially crucial in large-scale systems projects. Dashboards have been widely adopted on the project management side and on the development side [Johnson et al., 2005; Coman et al., 2009]. However, dashboards for requirements have not been as popular. According to [Pauwels et al., 2009], different domains have different needs with regard to dashboards. Thus, establishing criteria and design guidelines that address the needs of requirements measurement is one step towards building effective requirements dashboards. The attributes, levels, metrics, and meta-data in this study can serve as a guide for persons or organizations interested in implementing a requirements dashboard. Moreover, the requirements metrics in this paper can be used to complement and enhance current dashboards that focus on code metrics and/or project management metrics.

3.6.2 Implications for Research

To our knowledge, our study is the first of its kind to address the four elements of requirements measurement that we addressed in our study. As demonstrated in Section 3.2.5, the different elements have either been dealt with in piecemeal fashion, superficially or in unknown contexts. Thus, the exposition of all the elements in this paper is a significant contribution to the current literature on RE metrics and has several implications for research.

Research on Measurable Requirements Attributes. While our literature analysis reveals that the focus has been on requirements quality [Davis et al., 1993] (e.g., individual requirement size [Génova et al., 2013], complexity [Antinyan and Staron, 2017], consistency [Byun et al., 2014], etc.) and volatility [Zowghi and Nurmuliani, 2002; Loconsole and Börstler, 2005; Kulk and Verhoef, 2008; Ferreira et al., 2009; Thakurta and Ahlemann, 2010; Valerdi and Pena, 2015], our results show that, in addition to quality and volatility, size, growth, status and coverage are important attributes to measure in a systems project that aid in managing requirements and providing information to internal stakeholders to carry out their tasks (see Section 3.5.1). Thus, further research to better understand these attributes is required because a thorough understanding of a measured entity's attributes is essential for the derivation of accurate metrics [Briand et al., 1995]. Studies would investigate the requirement attributes in relation to other project variables (e.g., performance, cost, and schedule) and other requirements attributes (e.g., the relationship between requirements volatility and coverage) and define benchmarks, manage associated risks, and study their predictors. The literature on requirements volatility sets a good example for the comprehensive and holistic manner that other requirements attributes should be studied (see Section 3.2.1). In addition, further research is required to investigate other requirements attributes that need to be measured in order to facilitate the requirements management task and development processes in projects of different types (e.g., systems and software).

Research on Requirement Metric Levels. Due to the size and complexity of requirements in large systems projects, our results show that they exist at different levels, with each level requiring a set of metrics. Such results are not surprising as SE metrics in general have been defined to target different levels of software. However, we saw in Section 3.2.5 that such a categorization for RE metrics is nonexistent (See Table 3.1) although the categorization of requirements to enhance the documentation, understanding and management of requirements is recommended practice in general (e.g., functional and non-functional requirements) [Kotonya and Sommerville, 1998; IEEE, 2011]. In a series of empirical studies, Hess et al. [Gross and Doerr, 2012a,b; Hess et al., 2017] investigate the information needs from a software requirements specification (SRS) for different roles (e.g., architects and testers). They found that architects and testers want to view different information from the SRS. For example, architects wanted easy access to technical constraints and system interaction information while testers needed to view uses cases to carry out their test-related activities. Although the studies focused on the presentation of qualitative requirements data from the requirements specification (e.g., descriptions of non-functional requirements and descriptions of technical constraints, etc.), the different information needs imply that the different stakeholders would need different quantitative data (i.e., requirement measures) from the SRS as well. Our results further support this notion and, thus, are anticipated to promulgate further research that examines that different stakeholder needs with regard to quantitative requirements information at the different metric levels.

Moreover, requirements attributes are usually studied standalone and requirements levels are used merely to categorize and organize requirements. However, studying them together for the purpose of requirements measurement is unprecedented. Our results plant the seeds to further investigate this intersection of attributes and levels with regard to metrics, internal stakeholder concerns and information needs, and applicability of the different attribute-level combinations to different contexts.

Research on Requirement Metrics. The metrics presented in this paper have been defined within the context of one project and, thus, addressing the needs of that particular project. Moreover, an attribute can be measured through various metrics [Fenton and Biemen, 2015]. Thus, further research is needed to identify other requirements metrics that would aid the requirements management and development processes in different contexts, while maintaining a rigorous standard of metric definition and validation.

In addition, given that the defined metrics presented in this study were informed by internal stakeholder concerns at a certain point in time, they can be characterized as simple requirements metrics consisting of basic counts and comparisons. However, the metrics in this study lay the foundation for more complex requirements metrics that the internal stakeholders may have not considered and that would potentially be beneficial to them. For example, metrics that calculate the likelihood that a certain release, feature, or individual requirement will change based on volatility measures can be calculated using the volatility metrics presented here.

Finally, the validation processes we followed in this study are anticipated to set the scene for further empirical validation of the presented metrics and RE metrics in general in order for RE metrics to achieve the level of validation the SE metrics have achieved [Kitchenham, 2010].

3.7 Conclusions and Future Work

Metrics and measurement have been a focus of the SE community's attention since its early days and had proved effective in understanding, controlling and improving SE products and processes [Fenton and Biemen, 2015]. In requirements engineering, while the literature and practitioners generally recommend the usage of requirements metrics and attest to its many benefits (e.g., tracking progress, identifying gaps in the downstream deliverables, and managing requirements-related risks, etc.) [Costello and Liu, 1995; IEEE, 2011; Kratschmer, 2013], in reality RE metrics have been underused and underutilized despite the fertile ground that requirements and requirements meta-data provide for measurement [Gómez et al., 2008].

Applying requirements metrics becomes more pertinent in large-scale systems engineering projects where the large number of requirements, numerous processes and internal stakeholders (e.g., architects, developers, and product and safety managers) make it difficult to track and manage requirements and to provide the relevant requirements-driven information to the internal stakeholders who would use such information to carry out their respective process-related tasks. Our experience in a large-scale systems engineering project in the rail automation domain revealed that defining and applying requirements metrics is difficult due to: i) the lack of well-defined and validated metrics in the literature that can be readily used, and ii) the difficulty of defining requirements metrics due to lack of understanding of what to measure, at what metric levels, and the overhead associated with gathering and maintaining the required meta-data needed to apply the metrics.

We, thus, aimed in this paper to provide an empirically derived and validated requirements metric suite that would enable the tracking and control of requirements and requirements driven information and provide the relevant requirements driven information to internal stakeholders. We conducted an action research study in a large-scale project in the rail-automation domain that consisted of three sub-projects in which we investigated four research questions (Section 3.3.2) concerning the requirements metrics, measurable requirements attributes, metric levels, and requirements meta-data in a systems engineering project. The AR study resulted in defining 90 requirements metrics (Section 3.4.4) that measure five requirements attributes (Section 3.4.1) at four metric levels (Section 3.4.2) and that used nine requirements meta-data items (Section 3.4.3). We discuss the findings of each measurement element in this paper and validate the metrics theoretically and empirically for usefulness by applying them in the requirements management and development process of the project (Section 6.5).

Our validation results led us to several conclusions. First, that requirements and requirements meta-data can be utilized to define and apply useful requirements metrics that can be incorporated into the requirements management process and used throughout the development processes. Second, the requirements metrics did, in fact, facilitate the requirements management process and provided internal stakeholders with requirements-driven information that aided them in carrying out their development related tasks (Section 3.5.1). Such tasks included ensuring requirements-design coverage by architects, release planning by systems and products managers, safety requirements management by safety managers, and communication to upper management and clients. These observed uses are in line with those discussed in the literature [Davis et al., 1993; Costello and Liu, 1995; Wiegers, 2006; IEEE, 2011; Kratschmer, 2013]. Third, we found that different measures for the various attribute-level combinations (e.g., size at the baseline level and size at the release level) are used differently by internal stakeholders (Section 3.5.1). This finding echoes similar findings in the literature that have investigated information needs for different internal stakeholders. For example, testers and architects have different information needs with regard to the requirements document [Gross and Doerr, 2012a,b; Hess et al., 2017] and developers and managers need different code-centric information [Buse and Zimmermann, 2012].

Our next steps include automating the metric suite through a requirements dashboard and deploying the dashboard in the studied projects. The dashboard will allow us to investigate more closely the usage of the requirement metrics by the various internal stakeholders. Thus, we plan to further validate the findings of this study via a more controlled and quantitative validation method to assess whether the initial validation results still hold and by what degree.

Finally, our study was conducted in a large-scale project with a mature requirements engineering process that was receptive to the incorporation of metrics into its processes. However, the applicability and usefulness of such metrics in other contexts, particularly an agile one, is not yet known. Thus, we intend to validate the metrics in another contexts to investigate whether having a ready requirements metrics suite will improve the likelihood of adopting requirements metrics and whether the same uses and benefits are observed in that context.

References

[Antinyan and Staron, 2017] Antinyan, V. and Staron, M. (2017). Rendex: A method for automated reviews of textual requirements. *Journal of Systems and Software*, 131:63–77.

[Antinyan et al., 2016] Antinyan, V., Staron, M., Sandberg, A., and Hansson, J. (2016). Validating software measures using action research a method and industrial experiences. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE '16)*, pages 1–10, Limerick, Ireland. ACM.

[Basili et al., 1994] Basili, V., Caldiera, G., and Rombach, H. (1994). Goal question metric approach. *Encyclopedia of Software Engineering*, 1:98–102.

[Berenbach and Borotto, 2006] Berenbach, B. and Borotto, G. (2006). Metrics for model driven requirements development. In *28th International Conference on Software Engineering (ICSE 2006)*, pages 445–451, Shanghai, China. ACM.

[Boehm, 1991] Boehm, B. W. (1991). Software risk management: principles and practices. *IEEE Software*, (January):32–40.

[Briand et al., 1995] Briand, L., El Emam, K., and Morasca, S. (1995). Theoretical and empirical validation of software product measures. Technical report, International Software Engineering Network.

[Briand et al., 1996] Briand, L. C., Morasca, S., and Basili, V. R. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86.

[Buse and Zimmermann, 2012] Buse, R. P. L. and Zimmermann, T. (2012). Information needs for software development analytics. In *34th International Conference on Software Engineering (ICSE 2012)*, pages 987–996, Zurich, Switzerland. IEEE.

[Byun et al., 2014] Byun, J. W., Rhew, S. Y., Hwang, M. S., Sugumara, V., Park, S. Y., and Park, S. J. (2014). Metrics for measuring the consistencies of requirements with objectives and constraints. *Requirements Engineering*, 19(1):89–104.

[Carvalho et al., 2018] Carvalho, R. M., Andrade, R. M. d. C., and de Oliveira, K. M. (2018). AQUArIUM - A suite of software measures for HCI quality evaluation of ubiquitous mobile applications. *Journal of Systems and Software*, 136:101–136.

[Chemuturi, 2013] Chemuturi, M. (2013). *Requirements engineering and management for software development projects*. Springer.

[Chidamber and Kemerer, 1994] Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

[Cleland-Huang et al., 2012] Cleland-Huang, J., Gotel, O., and Zisman, A., editors (2012). *Software and systems traceability*. Springer.

[Coman et al., 2009] Coman, I. D., Sillitti, A., and Succi, G. (2009). A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*, pages 89–99, Vancouver, Canada. IEEE.

[Costello and Liu, 1995] Costello, R. J. and Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems Software*, 29:39–63.

[Davis et al., 1993] Davis, A., Overmeyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., and Theofanos, M. (1993). Identifying and measuring quality in a software requirements specification. In *Proceedings of the 1st International Software Metrics Symposium*, pages 141–152, Baltimore, MD, USA. IEEE.

[Davison et al., 2012] Davison, R. M., Martinsons, M. G., and Uo, C. X. J. (2012). The roles of theory in canonical action research. *MIS Quarterly*, 36(3):763–786.

[Easterbrook et al., 2008] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2008). Selecting empirical methods for software engineering research. In Shull, F., Singer, J., and Sjoberg, D. I., editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer.

[Ebert and Dumke, 2007] Ebert, C. and Dumke, R. (2007). *Software measurement: establish, extract, evaluate, execute*. Springer.

[Fenton and Biemen, 2015] Fenton, N. and Biemen, J. (2015). *Software metrics a rigorous and practical approach*. CRC Press, third edition.

[Ferreira, 2002] Ferreira, S. (2002). *Measuring the effects of requirements volatility on software development projects*. PhD thesis, Arizona State University.

[Ferreira et al., 2009] Ferreira, S., Collofello, J., Shunk, D., and Mackulak, G. (2009). Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation. *Journal of Systems and Software*, 82(10):1568–1577.

[Ferreira et al., 2011] Ferreira, S., Shunk, D., Collofello, J., Mackulak, G., and Dueck, A. (2011). Reducing the risk of requirements volatility: findings from an empirical survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 23:375–393.

[Génova et al., 2013] Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., and Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41.

[Gómez et al., 2008] Gómez, O., Oktaba, H., Piattini, M., and García, F. (2008). A systematic review measurement in software engineering: State-of-the-art in measures. In *Proceedings of the 3rd International Conference on Software and Data Technologies (ICSOFT'08)*, pages 165–176.

[Gotel et al., 2012] Gotel, O., Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., Grunbacher, P., and Antoniol, G. (2012). The quest for ubiquity: A roadmap for software and systems traceability research. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE'12)*, pages 71–80, Chicago, Illinois, USA. IEEE.

[Gotel and Finkelstein, 1994] Gotel, O. C. Z. and Finkelstein, A. C. W. (1994). An analysis of the requirements traceability problem. In *Proceedings of the 1st International Conference on Requirements Engineering (RE'94)*, pages 94–101, Colorado Springs, CO , USA. IEEE.

[Goti, 1998] Goti, J. C. (1998). Metrics for requirements management. Technical report, Rational Software Corporation.

[Gross and Doerr, 2012a] Gross, A. and Doerr, J. (2012a). What do software architects expect from requirements specifications? Results of initial explorative studies. In *1st IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks 2012)*, pages 41–45, Chicago, Illinois. IEEE.

[Gross and Doerr, 2012b] Gross, A. and Doerr, J. (2012b). What you need is what you get!: The vision of view-based requirements specifications. In *20th IEEE International Requirements Engineering Conference (RE'12)*, pages 171–180, Chicago, Illinois, USA. IEEE.

[Helmer, 1967] Helmer, O. (1967). Systematic use of expert opinions. Technical report, The RAND Corporation, Santa Monica, California.

[Hess et al., 2017] Hess, A., Doerr, J., and Seyff, N. (2017). How to make use of empirical knowledge about testers' information needs. In *25th IEEE International Requirements Engineering Conference Workshops (REW 2017)*, pages 327–330. IEEE.

[IEEE, 1989] IEEE (1989). IEEE standard dictionary of measures to produce reliable software. Technical report, IEEE.

[IEEE, 1992] IEEE (1992). IEEE 1061 standard for a software quality metrics methodology. Technical report, IEEE.

[IEEE, 2011] IEEE (2011). ISO/IEC/IEEE 29148 - Systems and software engineering — Life cycle processes — Requirements engineering. Technical report, IEEE.

[Jayatilleke and Lai, 2018] Jayatilleke, S. and Lai, R. (2018). A systematic review of requirements change management. *Information and Software Technology*, 93:163–185.

[Johnson et al., 2005] Johnson, P. M., Kou, H., Paulding, M., Zhang, Q., Kagawa, A., and Yamashita, T. (2005). Improving software development management through software project telemetry. *IEEE Software*, 22(4):76–85.

[Jones, 1996a] Jones, C. (1996a). *Software systems failure and success.* International Thomson Computer Press, Boston, MA.

[Jones, 1996b] Jones, C. (1996b). Strategies for managing requirements creep. *IEEE Computer*, 29(6).

[Jones, 2000] Jones, C. (2000). *Software assessments, benchmarks, and best practices.* Addison-Wesley.

[Kerzner, 2017] Kerzner, H. (2017). *Project management metrics, KPIs, and dashboards: A guide to measuring and monitoring project performance.* John and Wiley & Sons, third edition.

[Kitchenham, 1995] Kitchenham, B. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944.

[Kitchenham, 2010] Kitchenham, B. (2010). What's up with software metrics? - A preliminary mapping study. *Journal of Systems and Software*, 83(1):37–51.

[Kolde, 2004] Kolde, B. C. (2004). Basic metrics for requirements management. Technical Report March, Borland.

[Kotonya and Sommerville, 1998] Kotonya, G. and Sommerville, I. (1998). *Requirements engineering: Processes and techniques.* John Wiley, New York.

[Kratschmer, 2013] Kratschmer, T. (2013). Requirements and metrics. *Requirements Management Blog - IBM.* www.ibm.com/developerworks/community/blogs/requirementsmanagement/entry/requirements-metrics?lang=en.

[Kulk and Verhoef, 2008] Kulk, G. P. and Verhoef, C. (2008). Quantifying requirements volatility effects. *Science of Computer Programming*, 72(3):136–175.

[Lago et al., 2009] Lago, P., Muccini, H., and Van Vliet, H. (2009). A scoped approach to traceability management. *The Journal of Systems and Software*, 82:168–182.

[Lam et al., 1999] Lam, W., Loomes, M., and Shankararaman, V. (1999). Managing requirements change using metrics and action planning. In *Proceedings of the 3rd Euromicro Conference on Software Maintenance and Reengineering (CSMR'99)*, pages 122–128, Amsterdam, the Netherlands. IEEE.

[Leffingwell and Widrig, 2000] Leffingwell, D. and Widrig, D. (2000). *Managing software requirements: A unified approach*. Addison-Wesley.

[Loconsole, 2001] Loconsole, A. (2001). Measuring the requirements management key process area. In *Proceedings of the 12th European Software Control and Metrics Conference (ESCOM'01)*, pages 67–76, London, UK.

[Loconsole, 2003] Loconsole, A. (2003). Theoretical validation and case study of requirements management measures. Technical Report February, UMEA University.

[Loconsole and Börstler, 2005] Loconsole, A. and Börstler, J. (2005). An industrial case study on requirements volatility measures. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 249–256, Teipei, Taiwan. ACM.

[Loconsole and Borstler, 2007] Loconsole, A. and Borstler, J. (2007). A correlational study on four size measures as predictors of requirements volatility. Technical report, Umea University.

[Lormans and van Deursen, 2005] Lormans, M. and van Deursen, A. (2005). Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 47–52, Long Beach, California. ACM.

[Lormans and Van Deursen, 2006] Lormans, M. and Van Deursen, A. (2006). Can LSI help reconstructing requirements traceability in design and test? In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, pages 47–56. IEEE.

[Malaiya and Denton, 1999] Malaiya, Y. and Denton, J. (1999). Requirements volatility and defect density. In *Proceedings 10th International Symposium on Software Reliability Engineering*, pages 285–294, Boca Raton, FL, USA. IEEE.

[McGee and Greer, 2012] McGee, S. and Greer, D. (2012). Towards an understanding of the causes and effects of software requirements change: Two case studies. *Requirements Engineering*, 17(2):133–155.

[Morasca and Briand, 1997] Morasca, S. and Briand, L. (1997). Towards a theoretical framework for measuring software attributes. In *Proceedings of the 4th International Software Metrics Symposium*, pages 119–126, Albuquerque, NM, USA. IEEE.

[Moroney, 1968] Moroney, M. J. (1968). *Facts from figures*. Penguin Books, third edit edition.

[Nassar and Scandariato, 2017] Nassar, B. and Scandariato, R. (2017). Traceability metrics as early predictors of software defects? In *Proceedings of the International Conference on Software Architecture (ICSA'17)*, pages 235–238, Gothenburg, Sweden. IEEE.

[Nassar et al., 2016] Nassar, B., Shahrokni, A., and Scandariato, R. (2016). Traceability data in early development phases as an enabler for decision support. In *Proceedings of the Scientific Workshop (XP'16)*, pages 1–4, Edinburgh, Scotland. ACM.

[Noorwali et al., 2019] Noorwali, I., Madhavji, N. H., Arruda, D., and Ferrari, R. (2019). Towards a meta-model for requirements-driven information for internal stakeholders. In Knauss, E. and Goedicke, M., editors, *25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2019)*, volume 1, pages 262–278, Essen, Germany. Springer Nature.

[Nurmuliani et al., 2004] Nurmuliani, N., Zowghi, D., and Powell, S. (2004). Analysis of requirements volatility during software development life cycle. In *Proceedings of the Australian Software Engineering Conference*, pages 28–37, Melbourne, Victoria, Australia. IEEE.

[Offen and Jeffery, 1997] Offen, R. J. and Jeffery, R. (1997). Establishing software measurement programs. *IEEE Software*, 14(2):45–53.

[Park et al., 2010] Park, S., Maurer, F., Eberlein, A., and Fung, T.-S. (2010). Requirements attributes to predict requirements related defects. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, pages 42–56, Toronto, ON. ACM.

[Pauwels et al., 2009] Pauwels, K., Ambler, T., Clark, B. H., LaPointe, P., Reibstein, D., Skiera, B., Wierenga, B., and Wiesel, T. (2009). Dashboards as a service. *Journal of Service Research*, 12(2):175–189.

[Peterson and Wohlin, 2010] Peterson, K. and Wohlin, C. (2010). Software process improvement through the lean measurement (SPI-LEAM) method. *The Journal of Systems and Software*, 83:1275–1287.

[Runeson and Höst, 2009] Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164.

[Santos and Travassos, 2011] Santos, P. S. M. d. and Travassos, G. H. (2011). Action research can swing the balance in experimental software engineering. *Advances in Computers*, 83:205–276.

[Schneidewind, 1992] Schneidewind, N. F. (1992). Methodology for validating software metrics. *IEEE Transactions on Software Engineering*, 18(5):410–422.

[Selby, 2007] Selby, R. W. (2007). Data collection, analysis, and sharing strategies for enabling software measurement and model building. In Basili, V. R., Rombach, D., Schneider, K., Kitchenham, B., Pfahl, D., and Selby, R., editors, *Proceedings of the 2006 International Conference on*

Empirical Software Engineering Issues: Critical Assessment and Future Directions, volume LNCS 4336, pages 70–76. Springer.

[Selby, 2009] Selby, R. W. (2009). Analytics-driven dashboards enable leading indicators for requirements and designs of large-scale systems. *IEEE Software*, 26(1):41–49.

[Srinivasan and Devi, 2014] Srinivasan, K. P. and Devi, T. (2014). Software metrics validation methodologies in software engineering. *International Journal of Software Engineering & Applications (IJSEA)*, 5(6):87–102.

[Staron, 2019] Staron, M. (2019). Action research in software engineering: Metrics' research perspective (invited talk). In Catania, B., Královič, R., Nawrocki, J., and Pighizzini, G., editors, *45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2019: Theory and Practice of Computer Science)*, pages 39–49, Nový Smokovec, Slovakia. Springer.

[Susman and Evered, 1978] Susman, G. and Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative Science Quarterly*, 23(4):582–603.

[Thakurta and Ahlemann, 2010] Thakurta, R. and Ahlemann, F. (2010). Understanding requirements volatility in software projects - An empirical investigation of volatility awareness, management approaches and their applicability. In *Proceedings of the 43rd Annual Hawaii International Conference on System Sciences (HICSS'10)*, pages 1–10, Koloa, Kauai, Hawaii, USA. IEEE.

[Thorpe and Holt, 2008] Thorpe, R. and Holt, R., editors (2008). *The Sage dictionary of qualitative management research.* SAGE.

[Trudel and Abran, 2008] Trudel, S. and Abran, A. (2008). Improving quality of functional requirements by measuring their functional size. In Dumke, R., Braungarten, R., Buren, G., Abran, A., and Cuadrado-Gallego, J. J., editors, *Software Process and Product Measurement*, volume 5338 LNCS, pages 287–301. Springer.

[Valerdi and Pena, 2015] Valerdi, R. and Pena, M. (2015). Characterizing the impact of requirements volatility on systems engineering efforts. *Systems Engineering Journal*, 18(1):59–70.

[Wiegers, 2006] Wiegers, K. E. (2006). *More about software requirements: Thorny issues and practical advice.* Microsoft Press, Washington.

[Yousuf, 2007] Yousuf, M. I. (2007). Using experts' opinions through Delphi Technique - Practical assessment, research and evaluation. *Practical Assessment, Research and Evaluation*, 12(4).

[Zowghi and Nurmuliani, 2002] Zowghi, D. and Nurmuliani, N. (2002). A study of the impact of requirements volatility on software project performance. In *Proceedings of the 9th Asia-Pacific Software Engineering Conference (APSEC'02)*, pages 3–11, Queensland, Australia. IEEE.

Chapter 4

An Approach for Defining, Analyzing, and Organizing Requirements Metrics and Related Information [1]

4.1 Introduction

Measurement is an essential yet difficult process in software engineering (SE). It generally follows the establish-prepare-execute-evaluate measurement paradigm [IEEE, 2017; Ebert and Dumke, 2007] that consists of many tasks and sub-tasks. The longest and most complicated phase is the 'prepare' phase in which, among other tasks, the measurement strategy is identified, information needs are identified and prioritized, measures are selected and specified, and data collection, analysis, access and reporting procedures are defined [IEEE, 2017]. Many frameworks, tools and methods have been proposed to facilitate these tasks. For example, CMMI [CMMI Product Team, 2006] may be used to define the organization's measurement strategy while the Goal-Question-Metric (GQM) approach [Basili et al., 1994] aids in identifying and prioritizing information needs and selecting the measures that align with those needs.

In large systems projects, deriving and organizing requirements metrics and related information (e.g., meta-data items, measures, and metric labels) can be complicated and laborious due to such factors as: large volume of requirements; inconsistent requirements meta-data across sub-projects; and complexity in requirements baselines (that contain categories such as interface, hardware, software, each of which has its own set of metrics). In addition, the derived requirements metrics are numerous, often unstructured and unorga-

[1] A version of this paper was accepted for publication in the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'20). Publication to appear in May 2020.

nized, and can be difficult to assess with respect to completeness.

Existing measurement methods are aimed at either selecting and specifying a set of metrics that address certain project goals (e.g., GQM [Basili et al., 1994] and V-GQM [Olsson and Runeson, 2001]) or documenting metrics and measurement reports through templates [Goethert and Siviy, 2004; Bonelli et al., 2017]. For example, GQM [Basili et al., 1994] aids in documenting assessment or improvement goals and in deriving the questions and metrics that address these goals, all hierarchically represented. Meanwhile, measurement templates [Goethert and Siviy, 2004] help in recording data corresponding to the metrics.

However, once the hierarchy of goals-questions-metrics is identified and prior to gathering measures in templates, we are left with at least the following questions: *What requirements meta-data items (e.g., release number, status, and feature ID) do we need to apply the metrics? Are any metrics missing that may affect the investigation? How do we organize and structure these metrics for reporting?*

These questions are important because they impact the time needed to define, apply, and organize the measures for dissemination, quality of generated reports, completeness and consistency of the metrics, and completeness and consistency of the meta-data. In other words, these questions correspond to the structure that helps in organising and operationalizing the use of metrics in large, systems engineering projects. To our knowledge, the scientific literature does not contain such a structure that bridges the gap between intention (e.g., the GQM-like hierarchy) and use of metrics in actual projects.

In this paper, we show what the bridging structure is and how to use this structure through a 7-step process to derive and organize requirements metrics (Section 4.3). Specifically, in this process, GQM is first used to identify measurement goals, questions, and an initial set of corresponding metric descriptions. Requirements attributes (e.g., volatility, coverage, and growth) and levels (e.g, feature, release, and safety) are then identified for the initial set of metrics. We then identify all the possible attribute-level combinations (e.g., feature growth, and release volatility) and map the identified metrics onto the attribute-level combinations. The *metric gaps* are then identified and their missing metrics are derived. Finally, the meta-data (e.g., release number, feature ID, and safety relevancy) for each metric is identified. By the end of this process, we should have a complete set of requirements metrics at the identified attribute-level combinations with the meta-data items required to apply them.

By applying our approach on data from an industrial-scale rail automation systems project (Section 4.4.2), we discuss the observed benefits of its application (Section 4.4.3). The benefits included reduction of requirements measurement time, improved organization and structure of data, improved breadth of metrics, and improved completeness and consistency of meta-data across projects. We then discuss the implications of our work (Section 4.5.1) and its limitations (Section 4.5.2). Finally, we conclude the paper and discuss future work (Section 4.6).

4.2 Background and Related Work

In this section, we first lay the background for our work, which consists of: (i) the terminology used throughout this paper; namely, the four measurement components that our approach rests on: requirements attributes, levels, metrics, and meta-data (Section 4.2.1) and (ii) the general measurement process as defined by the ISO/IEEE standard [IEEE, 2017] and show where our approach fits within the process (Section 4.2.2). We then survey the related literature and discuss measurement frameworks and approaches (Section 4.2.3). Finally, we analyze the related work and highlight the research gap (Section 4.2.4).

4.2.1 Terminology

Requirement Attributes. An attribute is a property of an entity [Fenton and Biemen, 2015]. For instance, a person is an entity and height is one of the person's attributes. Thus, a requirement attribute is a property that a requirement or set of requirements (i.e., entity) possesses such as size, growth, volatility, quality, etc.

Requirement Levels. We use requirements levels to refer to the different, yet interrelated, categories according to which a set of requirements can be organized. For example, a set of requirements may be organized as a whole in requirement baselines (i.e., requirement document versions), which are further organized within the baseline according to features. Meaning, requirements exist at the baseline and feature levels.

Requirement Metrics. A "function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possess a given attribute" [IEEE, 1992]. A requirement metric, then, is a function whose inputs are requirement data and whose output is a single numerical value that can be interpreted as the degree to which requirements possess a given attribute. For example, *percentage of the number of additions, deletions, and modifications per baseline* is a metric that can have a numerical value (i.e., measure) of 30%, which indicates the degree the requirements baseline possesses volatility.

Requirement Meta-data. The IEEE 29148 standard for RE [IEEE, 2011] recommends that requirements have descriptive meta-data to help understand and manage the requirements. The meta-data are associated with requirements in a requirements repository and may include unique identification, dependency, risk, source, type, rationale, and difficulty to name a few.

4.2.2 The General Measurement Process

Figure 4.1 shows an overview of the measurement process, which follows an *establish, prepare, perform, evaluate* process [IEEE, 2017; Ebert and Dumke, 2007]. According to the ISO/IEEE standard for systems and software engineering measurement [IEEE, 2017], the *establish* phase involves accepting the requirements for measurement and assigning resources to prepare, execute, and evaluate the measurement process. The *prepare* phase consists of the largest number of tasks and, subsequently, requires the longest time and the most effort. Figure 4.1 depicts the tasks within the *prepare* phase. The *perform* phase involves integrating procedures for data collection, analysis, and reporting to the relevant processes, collecting, storing, and verifying the data, analyzing and developing information items, and recording the results and informing the measurement users of results. Finally, the *evaluate* phase consists of evaluating the information products (i.e., measures) and measurement process and identifying potential improvements. Figure 4.1 shows where our approach fits within the general measurement process in comparison with other frameworks, approaches and templates, which we will review in the following subsection.

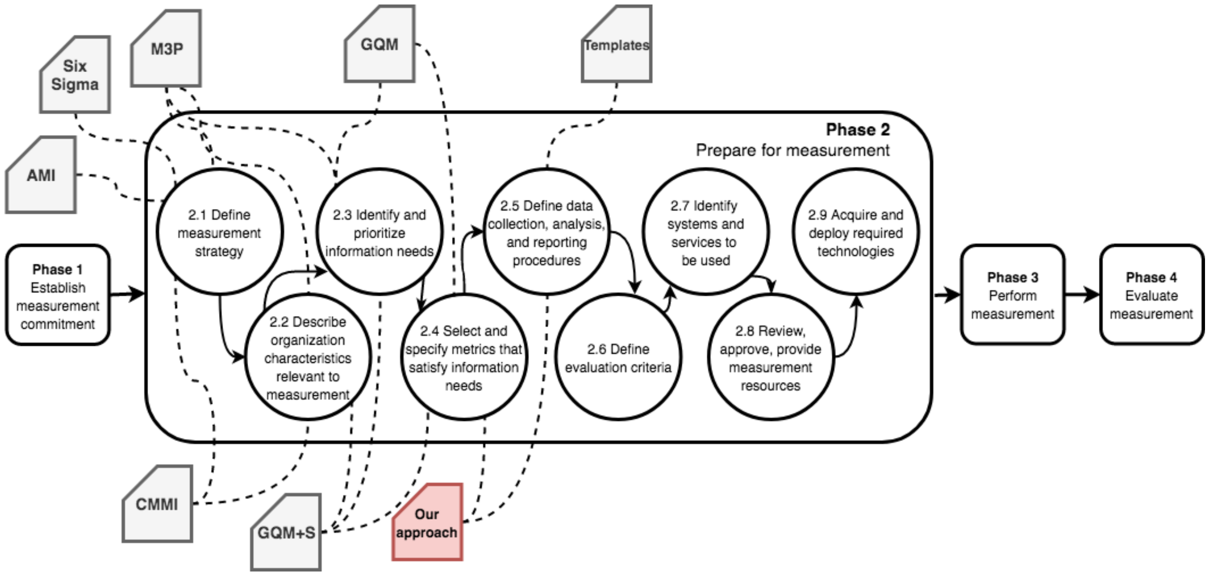

Six Sigma
M3P
GQM
Templates
AMI
Phase 2
Prepare for measurement
Phase 1 Establish measurement commitment
2.1 Define measurement strategy
2.2 Describe organization characteristics relevant to measurement
2.3 Identify and prioritize information needs
2.4 Select and specify metrics that satisfy information needs
2.5 Define data collection, analysis, and reporting procedures
2.6 Define evaluation criteria
2.7 Identify systems and services to be used
2.8 Review, approve, provide measurement resources
2.9 Acquire and deploy required technologies
Phase 3 Perform measurement
Phase 4 Evaluate measurement
CMMI
GQM+S
Our approach

Figure 4.1: The measurement process according to [IEEE, 2017].

4.2.3 Measurement Frameworks, Approaches, and Tools

A set of measurement frameworks, approaches, tools, and templates are required to carry out the tasks delineated in the measurement process described above. In this subsection, we discuss the relevant literature on such frameworks and approaches and situate them within the measurement process. The bulk of our discussion is dedicated to the approaches used for the *prepare* phase then more briefly the frameworks and approaches for the *perform* and *evaluate* phases. We do not discuss the *establish* phase because it mainly rests upon managerial decisions rather than on measurement approaches and methods.

As mentioned, the *prepare* phase consists of the largest number of tasks and requires the longest time and the most effort. Consequently, the literature is replete with methods, tools, and approaches that facilitate the tasks in this phase. Particularly, measurement frameworks and approaches that aid in defining the measurement strategy and describing organizational characteristics that are relevant to measurement (tasks 2.1 and 2.2 in Figure 4.1) have received a significant amount of attention. Such approaches and frameworks usually answer the *why* and *what* of measurement and can be used on the organizational, project, or process levels. For example, the Capability Maturity Model Integration (CMMI) is a process level improvement training and appraisal program [CMMI Product Team, 2006] that consists of a number of process areas, each process area is a cluster of related practices in that area that need to be implemented collectively to satisfy the goals considered important for making improvements in that area. *Measurement and Analysis* is one of the CMMI process areas consisting of practices and guidelines that echo the phases and activities of the ISO/IEEE standard: specifying measurement objectives, specifying measures, analysis techniques, data collection and reporting techniques, implementing analysis techniques and providing objective results.

The Model-Measure-Manage (M3p) paradigm [Offen and Jeffery, 1997] is another a measurement paradigm that consists of eight stages: understanding the business strategy, identifying business goals, strategies, and risks, determining critical success factors, defining specific software development goals, posing questions, identifying and defining measures, setting up the measurement program, and regularly reviewing the program. 'Application of metrics in industry' (AMI) [Rowe and Whitty, 1993] is yet another quantitative approach to software project management that consists of an 'assess-analyze-measure-improve' cycle. Similarly, Six Sigma [Tennant, 2001] is an approach that follows a 'define, measure, analyze, improve and control' methodology and employs statistical tools within its phases.

These higher-level paradigms employ lower level tools, templates, and approaches that aid in operationalizing lower-level tasks such as identifying users' information needs (task 2.3 in Figure 4.1) and selecting and specifying metrics (task 2.4 in Figure 4.1). An example is the GQM framework [Basili et al., 1994], which is one of the most popular techniques utilized within high-level measurement frameworks. For example, the M3P paradigm incorporates GQM as a measure selection technology and the AMI recommends using GQM in its 'analyze' phase.

The GQM approach aims to measure software in a meaningful way. The first step is to define a set of measurement goals tailored to the specific needs of the organization. The goals are refined into a set of quantifiable questions, for which, in turn, a specific set of metrics is specified. A more recent version of GQM, GQM+Strategies [Basili et al., 2014], has been proposed for aligning goals and strategies of an organization across different units through measurement. GQM+S consists of an 'organizational planning' part in which organizational goals and strategies are identified and a *controlling* part that utilizes the traditional GQM approach to achieve the organizational goals. Thus, GQM+S addresses tasks 2.2, 2.3, and 2.4 in the measurement process depicted in Figure 4.1.

Defining data collection, analysis, validation, documentation and reporting procedures (task 2.5 in Figure 4.1) requires grass-root measurement methods, templates, and tools to aid in executing these activities. Templates are usually used for data collection, documentation and reporting. For example, the Indicator Template for Measurement and Analysis [Goethert and Siviy, 2004] adds an intermediate step to GQM to assist in linking the questions to the measurement data that will be collected through a template that imposes the definitions of inputs (data elements) needed for measurement, and the data collection and reporting methods for each metric. Similarly, the ASM.br (Assistance for Software Measurement Based on Relationships) [Bonelli et al., 2017] is a template that allows specifying metrics by using a one-page form in which textual and graphical information is recorded, and the relationships between metrics and goals are explicitly presented. The template consists of the following information items: indicator (metric), business goal, measurement goal, information need, category, CMMI level and process area, measurement and analysis procedures, analysis procedure based on criteria, analysis procedure based on relationships, and indicator graphical representation.

Defining procedures for data analysis entails choosing the methods and techniques to be used for analyzing the gathered metrics. Such procedures could be as simple as using spreadsheets or more complex ones like machine learning techniques [Zhong et al., 2004] or statistical methods similar to the ones used in Six Sigma [Tennant, 2001].

For the *perform measurement* phase of the measurement process, using automated systems and tools is the most common approach to integrate the defined procedures for data collection, analysis and reporting from the *prepare for measurement* phase into software development the processes. Examples of such tools include Hackystat [Johnson, 2007], PRO Metrics (PROM) [Sillitti et al., 2003], and Empirical Project Monitor (EPM) [Ohira et al., 2004]. While the reporting and visualization capabilities of such tools can be limited, custom [Selby, 2009] and commercial dashboards and visualization software (e.g., Tableau) are used to visualize and communicate the resulting measures to the relevant stakeholders.

Finally, the measurement products and processes must be evaluated to identify potential problems and improvements (phase 4 of the measurement process in Figure 4.1). For example, Mendonca and Basili [Mendonça and Basili, 2000] propose an approach for improving exiting measurement frameworks that utilizes the GQM approach and an attribute focusing technique. The approach aims to better understand the ongoing measurement process within on organization or project and, in turn, explore the collected data in order to identify improvement recommendations for the measurement processes in place.

4.2.4 Analysis and Research Gap

While the above surveyed approaches and templates can be utilized for measurement in RE, they do not address the specific challenges we discussed in the introduction (i.e., unstructured and unorganized metrics, missing metrics, and missing meta-data). For example, let us assume that we used GQM to derive the set of metrics in Table 4.1, which is part of a larger set being used in the requirements management process. Upon derivation, the metrics lack a coherent structure as to which metrics belong with each other. Moreover, we have no method to assess whether there are missing metrics. The above surveyed approaches aid in either deriving the initial set of metrics or documenting the metrics in templates. To our knowledge, an approach that addresses the specific challenges of unstructured and unorganized metrics, missing metrics, and missing meta-data does not exist.

Table 4.1: Example requirements metrics derived using GQM.

Metric ID	**Metric Description**
M1	No. of requirements per baseline
M2	No. of modified requirements per feature baseline
M3	No. of safety critical requirements per baseline
M4	No. of requirements per release per baseline
M5	No. of deleted requirements per feature baseline

In this respect, our approach would be considered an intermediary step or *middleware* between metric derivation approaches such as GQM [Basili et al., 1994] and and using templates to document metrics [Goethert and Siviy, 2004; Bonelli et al., 2017]. Specifically, our approach aids in the selection of RE metrics (through GQM) and the analysis and reasoning about the metrics with respect to completeness, structure, and requirements meta-data collection (see Section 4.3).

4.3 The Approach

As mentioned earlier in Section 4.1, the purpose of our approach is to facilitate the requirements measurement process through: 1) deriving requirements metrics, 2) analyzing the metrics for completeness, 3) structuring and organizing the metrics, and 4) specifying the meta-data needed for the metrics. The approach is depicted in Figure 4.2.

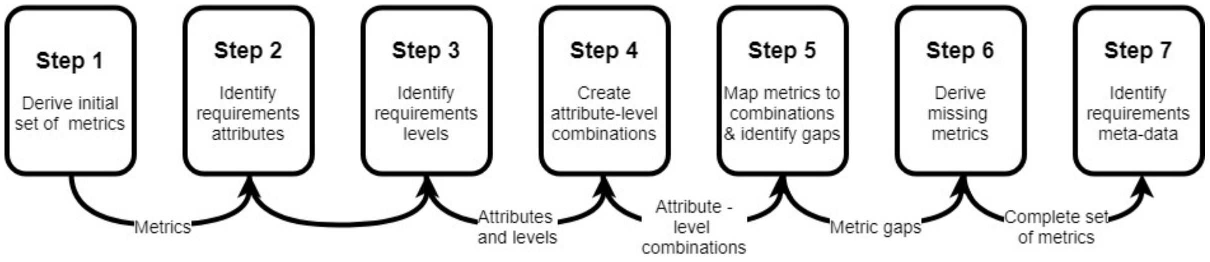

Step 1
Derive initial set of metrics
Step 2
Identify requirements attributes
Step 3
Identify requirements levels
Step 4
Create attribute-level combinations
Step 5
Map metrics to combinations & identify gaps
Step 6
Derive missing metrics
Step 7
Identify requirements meta-data
Metrics
Attributes and levels
Attribute - level combinations
Metric gaps
Complete set of metrics

Figure 4.2: The proposed 7-step approach for deriving, analyzing, and organizing requirements metrics.

The first step is executed using GQM [Basili et al., 1994] to derive an initial set of requirements metrics that address the internal stakeholders' goals and concerns. The second step is performed to identify the requirements attributes (e.g., size, volatility, and coverage) that the derived metrics are measuring while the third step identifies the requirements levels (e.g., baseline, release, and feature) at which the metrics exist. In the fourth step, we create all possible combinations of attributes and levels (e.g., *baseline X size* and *release X volatility*). In the fifth step, we map the derived metrics in Step 1 to the attribute-level combinations and identify the combinations that do not have associated metrics. Step 6 is performed to derive the metrics for the empty attribute-level combinations. Finally, in the seventh and last step, the requirements meta-data items for the complete set of metrics are identified. The approach combines the seven steps to tackle the following practical questions:

1. What requirements metrics will address the given internal stakeholders' concerns?

2. What requirements attributes are the metrics measuring?

3. How do we categorize and organize the derived metrics for archiving and reporting?

4. Are there any metrics missing that we are not aware of?

5. What meta-data do we need to gather in order to apply the metrics?

Step 1: Derive Initial Set of Metrics. The first step is to derive an initial set of metrics that address the internal stakeholders' concerns regarding the requirements. GQM is a suitable method to use for this step that ensures that the derived metrics in fact address internal stakeholder concerns and are not superfluous. The GQM approach consists of identifying the goals that the internal stakeholders of a project would like to achieve through the metrics. Following the goals, the operational questions that address those goals are subsequently identified and, finally, the metrics that answer the respective questions are derived. Thus, the output of the Step 1 is an initial set of requirements metrics. Table 4.2 shows an example of a goal and its respective questions and requirement metrics.

Table 4.2: An example of using GQM to derive requirements metrics for Step 1.

Goal		**Questions**	**Metrics**
Purpose:	Monitor	Q1. What is the overall state of requirements-design coverage?	M1. No. of requirements that are covered by design objects per baseline
Issue:	the status of		
Object:	requirements-design links	Q2. What is the state of requirements-design coverage for release X?	M2. No. of requirements in latest baseline that are covered by design and are assigned to release X
Viewpoint:	from the system's manager's viewpoint		

Step 2: Identify Requirements Attributes. Depending on how the goals and questions are formulated, the corresponding metrics can be derived without knowing the requirement attribute we are measuring. For example, it is not clear what requirement attribute M1 and M2 in Table 4.2 are measuring. Is it requirements status or requirements coverage? Thus, after the initial set of requirement metrics have been derived using GQM, we perform a first round of analysis of the metrics to answer the question: what requirement attribute is each metric is measuring? This step aids in acquiring a clear and unambiguous understanding of the requirements attributes being measured, which is essential for accurate measurements [Briand et al., 1996] and for reasoning about the derived metrics. For example, when applying Step 2 we realize that all the derived metrics from Step 1 are measuring *requirements volatility*, then we can begin reasoning whether we need other metrics that measure other attributes such as coverage, size, and creep, etc. Thus, we begin addressing the issue of missing metrics. In addition, this step begins giving the derived metrics a structure.

The output of Step 2 is a list of requirements attributes. The 'Requirement Metric' and

'Requirement Attribute' columns in Table 4.3 show a sample of requirements metrics and their corresponding attributes.

Table 4.3: Example requirements metrics, attributes, and levels.

Metric ID	**Requirement Metric**	**Requirement Attribute**	**Requirement Level**
M1	No. of requirements in latest baseline that are covered by design	Coverage	Baseline
M2	No. of requirements in latest baseline that are covered by design and are assigned to release X	Coverage	Release
M3	No. of requirements per feature per baseline	Size	Feature
M4	No. of added, deleted and modified requirements per baseline	Volatility	Baseline

Step 3: Identify Requirements Levels. Similar to the way requirements are organized, whether implicitly or explicitly, according to different categories or levels (e.g., functional and non-functional, features and releases), the derived requirements metrics will also exist at different requirements metric levels. Thus this step is concerned with answering the question: what level of requirements is the derived metric concerned with? The identification of requirements metric *levels* gives further structure to the derived metrics and allows us to reason about the breadth of metrics.

One way to identify the metric levels if they cannot be readily identified from the requirements documents, is to phrase the metrics in the form of M3 and M4 in Table 4.3 where we use 'per' followed by on object such as baseline and feature. Thus, the object of the first 'per' is a requirements level. In M3 and M4, it is easy to identify the levels (feature and baseline). However, due to the different wording of M1 and M2, it is not clear what the levels are. Thus, if we rephrase M1 to the following form: *number of requirements that have in-links from design objects per latest baseline*, we know that the requirement level is baseline. Similarly, M2 can be rephrased to the following form: *Number of requirements that have in-links from design objects per release per baseline.* Thus, the requirement level is release. Similarly, if we had a metric that measured *number of words per requirement* as a measure of an individual requirement's size [Antinyan and Staron, 2017], then we can say that this metric is at the individual requirement level.

Step 4: Create Attribute-Level Combinations. Once we identified the metrics' attributes and levels separately in steps 2 and 3, we create all possible combinations of the attributes and levels. It is important that we create all possible attribute-level combinations regardless of whether they have associated metrics. For example, we can create the following four combinations from the attributes and levels in Table 4.3 that have corresponding metrics: baseline coverage, release coverage, feature size, baseline volatility. However, Figure 4.3 shows all the nine possible attribute-level combinations that can be derived from Table 4.3. This step sets the scene for the next step in which we reason about missing metrics.

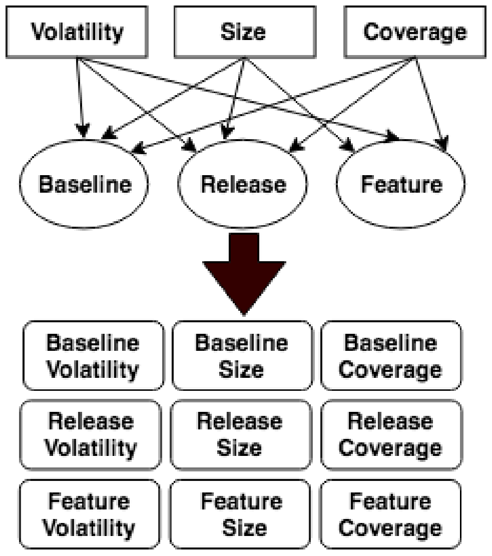


Figure 4.3: An example of attribute-level combinations.

Step 5: Map Metrics to Combinations and Identify Gaps. This step consists of mapping the metrics derived in step 1 to the relevant attribute-level combinations identified in step 4. This step is unnecessary if the metrics, attributes and levels identified in steps 1, 2, and 3 have already been tabulated together since the beginning of the process. However, if they are in separate files, then this step dictates creating a matrix consisting of all the attribute-level combinations and mapping the metrics onto the combinations. The matrix should also include the additional, empty combinations identified in Step 4, which would yield a table similar to Table 4.4. Thus, the new matrix highlights the attribute-level combinations that do not have corresponding metrics (i.e., metric gaps).

Table 4.4: Metrics mapped onto attribute-level combinations.

Metric ID	**Requirement Metric**	**Requirement Attribute**	**Requirement Level**
M1	No. of requirements in latest baseline that are covered by design	Coverage	Baseline
		Coverage	Feature
M2	No. of requirements in latest baseline that are covered by design and are assigned to release X	Coverage	Release
		Size	Baseline
M3	No. of requirements per feature per baseline	Size	Feature
		Size	Release
M4	No. of added, deleted and modified requirements per baseline	Volatility	Baseline
		Volatility	Feature
		Volatility	Release

Step 6: Derive Missing Metrics. In this step, we define the metrics for the empty attribute-level combinations. For example, for *feature X volatility* in Table 4.4, we can define *M5: The total number of added, deleted and modified requirements per feature per baseline.*

Step 7: Identify Requirements Meta-Data. Once we have defined the complete set of metrics, we identify the requirements meta-data items needed to apply the metrics. For example, the meta-data items needed for M2 are unique requirement ID, release number, in-links from design, requirement baseline number. Table 4.5 shows a an example of the meta-data that would be maintained for each requirement in order to apply the metrics identified in Step 6.

Table 4.5: Example requirements meta-data.

Baseline 3.1				
Req. ID	Req. Text	Release	In-Links	Feature
001	*******	3	****/****/ object12	External Interface

4.4 Applying the Approach in Practice

The most common and successful validation method for a software engineering approach or procedure [Shaw, 2003] is validation through an example based on a real-life scenario. Such validation can be accomplished in a variety of ways including case studies, experiments, or action research [Easterbrook et al., 2008]. In this subsection, we report our experience in applying our approach on data from a real-life industrial setting. Particularly, a large-scale rail automation systems project that consists of three sub-projects.

Initially, we conducted an action research (AR) study to derive and evaluate a set of requirements metrics to be incorporated into the requirements management and software development processes. However, we faced the challenges discussed in Section 4.1 during the study. Thus, the approach emerged as a by-product of the AR study to address those challenges. We then applied the proposed approach within the three sub-projects. We note that while the complete results from the AR study (i.e., requirements metrics) are not reported in this paper, we use a subset of the derived metrics to demonstrate the application of our approach.

In the following subsections, we briefly describe the project context and the AR study (Section 4.4.1), our experience with applying the approach (Section 4.4.2), and then discuss the observed benefits (Section 4.4.3).

4.4.1 The Projects and AR Study

The project in which we conducted the AR study is a large-scale rail automation project in a multi-national company in the United States. The overall project (i.e., program) consisted of many sub-projects, each sub-project consisted of a product that had its own set of requirements, architecture design, test cases, and engineering team. We were directly involved with three of the sub-projects. Table 4.6 shows a breakdown of project duration, number of requirements specification documents (baselines), number of requirements, and number of safety requirements per project. The organization used IBM Rational DOORS as their requirements management tool. Thus, each project's requirements were stored in its own DOORS database and identified with its own set of meta-data.

Table 4.6: Descriptive statistics of the projects.

Project	Project Duration	No. Req. Baselines	No. Reqs.	No. Safety Reqs.	No. Design Baselines	No. Design Objects	No. Test Cases
P1	73 months	54	1790	N/A	23	472	2111
P2	36 months	30	2285	N/A	4	380	N/A
P3	45 months	51	2389	923	28	827	2045

The goal of the AR study was to derive a set of requirements metrics for each project. The AR study followed an iterative process [Susman and Evered, 1978] in which the researcher, in collaboration with the industrial partners, identified the internal stakeholder needs with regard to the requirement metrics and the corresponding metric descriptions using GQM [Basili et al., 1994]. We opted to use GQM and not GQM+S because we were not concerned with organizational or project strategy.

4.4.2 Applying the Approach

In the following paragraphs, we describe in detail the application of our approach within Project 3 from Table 4.6. We note, however, that the approach was similarly applied to three other projects as well.

Step 1: Derive Initial Set of Metrics. Using GQM and in collaboration with the internal stakeholders, we derived an initial set of 41 requirements metrics. Table 4.7 consists of the goals, questions and titles of the associated metrics that we initially derived. Due to space restrictions, we do not list all the metric definitions. However, Table 4.8 shows the metric definitions for a subset of the metrics in Table 4.7.

Step 2: Identify Requirements Attributes. To execute this step, we identified the requirement attribute each metric is measuring as shown in Table 4.8 in the *Requirement At-*

Table 4.7: The initial set of requirements metrics using GQM.

Goal	Question	Metrics
G1. Monitor status of requirements-design coverage, requirements-test coverage.	Q1. What is the status of requirements-design coverage?	M1, M2, M3, M4
	Q2. What is the status of requirements–test coverage?	M5, M6, M7, M8
G2. Monitor growth and volatility of requirements.	Q3. What is the growth of requirements over time?	M9, M10, M11
	Q4. What is the volatility of requirements over time?	M12, M13, M14, M15, M16, M17, M18, M19, M20, M21, M22, M23, M24, M25, M26, M27
G3. Manage release planning of requirements.	Q5. What is the current state of allocations of requirements to releases?	M28, M29, M30, M31, M32, M33
G4. Monitor distribution and growth of safety requirements	Q6. What is the current distribution of safety requirements in latest baseline?	M34, M35, M36, M37, M38, M39, M40, M41

tribute column. The subset of metrics shown in Table 4.8 is representative of the the requirements attributes we initially identified: size, coverage, and volatility.

We note how the goals and questions do not necessarily lead to correct identification of attributes. For example, Q3 in Table 4.7 is concerned with the growth of requirements over time. However, the derived metrics (M9, M10) are in fact measuring *size*, but because when deriving the metrics, we envisioned that the measures will be visualized in a way that depicts requirements growth over time, the amount of growth in of itself is not being measured but the size of requirements over time. This led to the identification of the correct metrics for growth (M42, M43) in Table 4.8. Thus, at the end of this step we have added two metrics and a requirement attribute (growth).

Step 3: Identify Requirements Levels. We identified each metric's level according to the procedure described in Section 4.3. At the end of this step, we had four requirement metric levels: baseline, feature, release, and safety. Table 4.8 shows each metric's level in the *Requirement Level* column.

Step 4: Create Attribute-Level Combinations. From the identified attributes and levels in Table 4.8, we created all the possible attribute-level combinations. Because we have four attributes and four levels, we had 16 unique attribute-level combinations as identified in Table 4.9 in the *Attribute* and *Level* columns.

Table 4.8: A subset of the derived requirements metrics for project 3.

Metric ID	Requirement Metric	Requirement Attribute	Requirement Level
M1	No. of requirements that have in-links from design objects per baseline	Coverage	Baseline
M5	No. of requirements in latest baseline that have in-links from test cases	Coverage	Baseline
M9	No. of requirements per baseline	Size	Baseline
M10	No. of requirements per feature per baseline	Size	Feature
M12	No. of added requirements per baseline	Volatility	Baseline
M13	No. of deleted requirements per baseline	Volatility	Baseline
M14	No. of modified requirements per baseline	Volatility	Baseline
M15	No. of added, deleted and modified requirements per baseline	Volatility	Baseline
M16	No. of added requirements per feature per baseline	Volatility	Feature
M17	No. of deleted requirements per feature baseline	Volatility	Feature
M18	No. of modified requirements per feature baseline	Volatility	Feature
M19	No. of added, deleted and modified requirements per feature per baseline	Volatility	Feature
M29	No. of requirements per release per baseline	Size	Release
M30	Percentage of requirements per release per baseline	Size	Release
M34	No. of safety critical requirements per baseline	Size	Safety
M42	Difference between requirements size for baselines X and Y	Growth	Baseline
M43	Difference between requirements size for feature Z in baselines X and Y	Growth	Feature

Step 5: Map Metrics to Combinations and Identify Gaps. We map the metrics listed in Table 4.7 to the identified attribute-level combinations as depicted in Table 4.9. We can now identify the following metric gaps: *coverage X feature, coverage X release, coverage X safety, volatility X release, volatility X safety, growth X release, growth X safety*. Moreover, it is possible to detect missing metrics for the attribute-level combinations that *have* metrics by comparing the number of metrics for each combination. For example, size metrics on the release level (M29, M30) consist of an absolute and relative measure. However, size metrics on the feature (M10) and safety (M34) levels consist of absolute measures only.

Step 6: Derive Missing Metrics. The result of this step was an identification of 46 additional metrics that were incorporated into the overall metric set. Due to space restrictions, we do not include all the metrics that we derived upon identifying the metric gaps. However, the metrics in red in Table 4.10 show the new metrics and we give some examples here:

- **M52:** No. of requirements with in-links from test cases per feature per baseline.
- **M58:** No. of requirements with in-links from test cases per safety requirement category per baseline.
- **M72:** Percentage of modified requirements per release per baseline.
- **M87:** Difference between requirements size for release Z in baselines X and Y.

Table 4.9: All possible attribute-level combinations and mapping of metrics.

Attribute	**Level**	**Metrics**
Coverage	Baseline	M1, M2, M3, M5, M6, M7, M8
Coverage	Feature	
Coverage	Release	
Coverage	Safety	
Size	Baseline	M9
Size	Feature	M10, M11
Size	Release	M28, M29
Size	Safety	M34, M35, M36, M37, M38, M39, M40, M41
Volatility	Baseline	M12, M13, M14, M15, M16, M17, M18, M19
Volatility	Feature	M20, M21, M22, M23, M24, M25, M26, M27
Volatility	Release	
Volatility	Safety	
Growth	Baseline	M42
Growth	Feature	M43
Growth	Release	
Growth	Safety	

Table 4.10: Identification of missing metrics from Step 6.

Note: We chose not merge Tables 4.9 and 4.10 in order to highlight the *metrics gaps* in Table 4.9.

Attribute	**Level**	**Metrics**
Coverage	Baseline	M1, M2, M3, M4, M5, M6, M7, M8
Coverage	Feature	M44, M45, M46, M47,M49, M50, M51, M52
Coverage	Release	M53, M54, M55, M56,M57, M58, M59, M60
Coverage	Safety	M61, M62, M63, M64,M65, M66, M67, M68
Size	Baseline	M9
Size	Feature	M10, M11
Size	Release	M28, M29
Size	Safety	M34, M35, M36, M37, M38, M39, M40, M41
Volatility	Baseline	M12, M13, M14, M15, M16, M17, M18, M19
Volatility	Feature	M20, M21, M22, M23, M24, M25, M26, M27
Volatility	Release	M69, M70, M71, M72,M73, M74, M75, M76
Volatility	Safety	M77, M78, M79, M80,M81, M82, M83, M84
Growth	Baseline	M42, M85
Growth	Feature	M43, M86
Growth	Release	M87, M88
Growth	Safety	M89, M90

Step 7: Identify Requirements Meta-Data. Based on the final set of metrics we identify the meta-data needed to calculate each metric. The set of unique requirements meta-data items we identified as a result of identifying the meta-data items for each of the 90 metrics were: Requirement ID, Requirement type, Requirement feature ID, Requirement text,

Requirement release number, Safety requirement type, Out-links from requirements to external artifacts, In-Links to requirements. As an example, M1 from Table 4.8 would require an *out-links from requirements to external artifacts* meta-data item which we call *ReqOutlinks* for illustration purposes. Thus, the formula for M1 would be: *count if ReqOutlinks* $\neq$ *NULL*

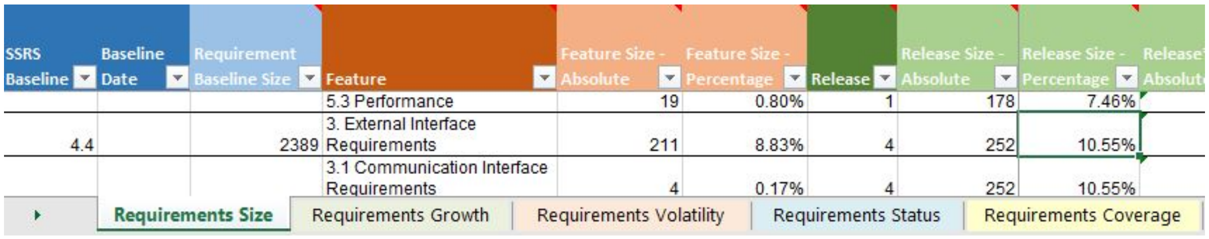
The meta-data items were necessary for ensuring that all meta-data items were consistent across projects and applying the metrics. This, in turn, facilitated the measurement procedure.

4.4.3 Observed Benefits

After illustrating the application of the approach in one of the rail automation projects, we discuss the overall benefits we observed from applying the approach to all the three projects listed in Table 4.6.

Metric Breadth. While GQM allows the identification of an initial set of metrics according to a project's goals, which, in turn, address the stakeholders' information needs, our experience with large systems projects that involve many internal stakeholders has shown further concerns with regard to the requirements metrics are identified upon having an initial set of metrics, which prompts further metric derivation. For example, as seen in Table 4.9, the initial set of metrics measured the design and test coverage of requirements for a requirements *baseline*. Upon implementing the metrics, an architect requested measures of requirements coverage per *feature*, for which we derived further metrics. However, our approach allowed us to derived the coverage metrics on the *release* and *safety* levels as well (see Table 4.10), which were also used by different internal stakeholders. Thus, our approach improves the breadth of the derived metrics by identifying the *metric gaps* and, subsequently, deriving the associated metrics. Because the approach identifies the metric gaps by analyzing the attributes and levels of the initial set of metrics that were derived using GQM and which are based on the the project's information needs, the missing metrics will likely also address measurement needs that the internal stakeholders were not cognizant of.

Organization of Data. Prior to using the approach and upon deriving the initial set of metrics in the AR study (see Section 4.4.1), the measures were documented in spreadsheets in an unorganized manner where metrics lacked accurate labels and unrelated metrics were grouped together. The identification of attributes and levels in our approach served as a *template*, which allowed us to structure measures in an organized and consistent format across projects. Figure 4.4 shows a snapshot from the requirements metric report for Project P3 in Table 4.6 in which the measures are organized according to requirements attributes (size, growth, volatility, status, coverage) and levels (baseline, feature, release).



SSRS Baseline	Baseline Date	Requirement Baseline Size	Feature	Feature Size - Absolute	Feature Size - Percentage	Release	Release Size - Absolute	Release Size - Percentage	Release Absolut
			5.3 Performance	19	0.80%	1	178	7.46%	
4.4		2389	3. External Interface Requirements	211	8.83%	4	252	10.55%	
			3.1 Communication Interface Requirements	4	0.17%	4	252	10.55%	

Requirements Size | Requirements Growth | Requirements Volatility | Requirements Status | Requirements Coverage

Figure 4.4: Requirements metric report organized according to attributes and levels.

Completeness and Consistency of Requirements Meta-Data. Initially, we adopted a tedious trial and error approach in which we analyzed each project's requirement meta-data to check whether the derived metrics can be applied to that particular project given the available meta-data. The requirements meta-data were incomplete (e.g., missing *release* meta-data) and inconsistent (i.e., different meta-data labels used such as *in-links* or *design links*) across projects. However, identifying the requirements meta-data upfront aided in assessing the completeness and consistency of the requirements meta-data in the databases across projects early in the measurement process. Moreover, the list of identified requirements meta-data items was incorporated into the requirements management plan to be enforced in future projects in order to facilitate the requirements measurement process.

Measurement Time and Effort. The application of the approach resulted in a reduction of the time and effort expended on the requirements measurement process, which was enabled in two ways. First, our experience in deriving metrics in a large systems project with numerous internal stakeholders showed that missing metrics are identified slowly and incrementally through feedback from the stakeholders as they elucidate their needs. The use of our approach reduces the time required for this process by identifying the metrics gaps and deriving the missing metrics through the upfront analysis and reasoning about the metrics. This enables the preemptive derivation of metrics that may be requested later on and which, in turn, reduces the time spent on measurement.

Second, the reuse of the identified attributes, levels, metrics, and meta-data items aided in reducing the time and effort needed to derive, analyze and organize a set of requirements metrics for each project. Thus, when a fourth new project was added, we simply reused the results of our approach from the previous three projects (Table 4.6). However, this reuse does not prevent reexamining the needs of a project and, subsequently, reapplying the approach to derive further metrics that address the newly identified needs.

4.5 Discussion

In this section, we first discuss the implications of our approach for RE management and measurement, requirements management tools, RE dashboards, and studies on RE measurement and then its limitations.

4.5.1 Implications

Requirements Management and Measurement Processes

The centrality of the four measurement elements (i.e., attributes, levels, metrics, and meta-data) in our approach could encourage requirements personnel to consciously consider the definition of the elements during the requirements management process. For instance, given that defining requirements meta-data is already an integral part of the requirements management process [Wiegers, 2006], requirements engineers can now define the requirements meta-data with requirements measurement in mind. For example, if the project intends to measure requirements volatility and coverage at the baseline and feature levels, then the requirements engineer would define the requirements meta-data that would facilitate the measurement of these attributes and levels later in the requirements engineering process. This, in turn, allows for seamless and easier application of our approach later in the requirements management process.

Existing Requirements Management Tools

While existing requirements management tools (e.g., Rational DOORS, Jama, and ReqSuite) allow the identification of requirements meta-data and derivation of some measures in relation to requirements (e.g., total no. of requirements, no. of requirements in progress), they do not support functionality for advanced requirement metric derivation, reasoning about metric completeness and consistency, and structuring metrics into related clusters. Thus, the requirements measurement process is carried out as an external process, which requires significant added time and effort [Costello and Liu, 1995]. The delineation of key requirements measurement elements and the steps to utilize them in our approach open up possibilities for advanced RE tool-features. For example, *attribute-level* combinations (e.g., *baseline X volatility*, *feature X coverage* —see Section 4.4.2 for examples) can be automatically generated, thereby saving effort and ensuring quality. Further, the meta-data items required for each metric can be selected from the list of defined meta-data in the requirements database and, based on the selected meta-data items, queries could be created to calculate measures.

RE Dashboards

Dashboards that gather, analyze, calculate, and present measures are commonly used in SE. Such dashboards have always targeted the project management [Kerzner, 2017] and development [Johnson, 2007] phases. However, different domains have different needs with regard to dashboards [Pauwels et al., 2009]. Whether on top of existing requirements management tools or as standalone dashboards, our approach may have practical implications on requirements dashboards in at least two ways. First, our approach could provide guidance to designing and developing requirements dashboards. For example, the concepts of attributes and levels can inform the presentation of the measures in the dashboard. Specifically, pages can be organized according to attributes (e.g., size, growth, and volatility, etc.) and each page can organize the measures according to levels (e.g., baseline, feature,and release, etc.) In addition, the dashboard can be designed to ensure that each metric is associated with an attribute and level to avoid 'stray metrics'. Second, the approach could be integrated as one of the dashboard's functionalities to enable the derivation, analysis and organization of metrics. For example, the dashboard could provide 'intelligent' recommendations for metrics based on defined attributes and levels. Thus, the dashboard would become more intelligent as more attributes, levels, and metrics are defined over time.

Further Studies on RE Measurement

While measurement approaches, such as GQM [Basili et al., 1994], have been widely adopted and have proven to be extremely effective, the plethora of methods and approaches that extend and build upon GQM suggest that GQM overlooks the needs of certain domains, processes, and/or stakeholders [Goethert and Siviy, 2004; Berander and Jönsson, 2006]. Our approach also addresses the specific challenges that have emerged in deriving and organizing RE metrics in large systems projects, which we believe is a first of its kind. Because our approach addresses specific challenges in the systems domain, it would be interesting to investigate other RE measurement methods that can be created in other domains. In addition, the applicability of our approach to other contexts (e.g., agile) can be further explored.

4.5.2 Limitations

This investigation was conducted in a large systems engineering company with well-established requirements management and documentation procedures and numerous internal stakeholders. Thus, the use of our approach can be said to be limited to such an environment. The applicability of our approach in other development environments (e.g., agile or planned agile) in which requirement documentation is minimal may be limited.

In addition, the approach assumes the existence of a large set of requirements in which requirements are organized according to different levels (e.g., features, releases, and baselines) and, thus, metric levels can be identified. However, in other process paradigms (e.g., agile and iterative, etc.), equivalent notions need to be identified similar to those in our approach.

Finally, without tool support, the application of our approach becomes tedious when the number of attributes, levels, metrics, and meta-data are large. Thus, incorporating the approach and its elements into requirements management tools, as discussed in Section 4.5.1, would facilitate its application in contexts where the number of metrics, attributes, levels, and meta-data become unmanageable manually.

4.6 Conclusions and Future Work

Requirements measurement in a systems engineering context is a complex task due to the existence of multiple projects with large sets of requirements with various categories (e.g., baseline, features, and releases, etc.), various internal stakeholders and their information needs, and inconsistent requirements meta-data across projects, to name a few. Thus, requirements metrics end up being: large in number, replicated with unintended variations, ill-structured and disorganized, and incomplete. Existing measurement approaches and methods do not address such requirements measurement concerns (Section 4.2.3).

In this paper, we propose an approach that aims to bridge the gap between the use of GQM to select and specify metrics that satisfy the needs of the stakeholders and the use of templates to document and report the measures. Particularly, the method aims to derive RE metrics, analyze them for completeness, structure and organize the metrics, and specify the meta-data needed for the metrics. The approach utilizes the GQM approach and consists of seven steps that rely on four measurement elements: requirements attributes, levels, metrics, and meta-data (Section 4.3). The approach aids in improving metric breadth, organizing measures, improving completeness and consistency of requirements meta-data, and reducing measurement time and effort (Section 4.4.3). We demonstrate the application of the approach to a real-life systems project from the rail automation domain (Section 4.4.2) and discuss its observed benefits. The approach is anticipated to have implication for requirements management measurement processes, requirements management tools, RE dashboards, and studies on RE measurement (Section 4.5.1). Future work rests in applying the approach in different contexts to strengthen its validity and providing tool support to improve its usability.

References

[Antinyan and Staron, 2017] Antinyan, V. and Staron, M. (2017). Rendex: A method for automated reviews of textual requirements. *Journal of Systems and Software*, 131:63–77.

[Basili et al., 1994] Basili, V., Caldiera, G., and Rombach, H. (1994). Goal question metric approach. *Encyclopedia of Software Engineering*, 1:98–102.

[Basili et al., 2014] Basili, V., Trendowicz, A., Kowalczyk, M., Heidrich, J., Seaman, C., Münch, J., and Rombach, D. (2014). *Aligning organizations through measurement*. Springer.

[Berander and Jönsson, 2006] Berander, P. and Jönsson, P. (2006). A goal question metric based approach for efficient measurement framework definition. In *Proceedings of the ACM/IEEE international symposium on Empirical software engineering (ISESE'06)*, pages 316–325, Rio de Janeiro, Brazil. ACM.

[Bonelli et al., 2017] Bonelli, S., Santos, G., and Barcellos, M. P. (2017). ASM.br: A template for specifying indicators. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, Karlskrona, Sweden.

[Briand et al., 1996] Briand, L. C., Morasca, S., and Basili, V. R. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86.

[CMMI Product Team, 2006] CMMI Product Team (2006). CMMI for development, Version 1.2. Technical report, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA.

[Costello and Liu, 1995] Costello, R. J. and Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems Software*, 29:39–63.

[Easterbrook et al., 2008] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2008). Selecting empirical methods for software engineering research. In Shull, F., Singer, J., and Sjoberg, D. I., editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer.

[Ebert and Dumke, 2007] Ebert, C. and Dumke, R. (2007). *Software measurement: establish, extract, evaluate, execute*. Springer.

[Fenton and Biemen, 2015] Fenton, N. and Biemen, J. (2015). *Software metrics a rigorous and practical approach*. CRC Press, third edition.

[Goethert and Siviy, 2004] Goethert, W. and Siviy, J. (2004). Applications of the indicator template for measurement and analysis. Technical report, Carnegie Mellon University.

[IEEE, 1992] IEEE (1992). IEEE 1061 standard for a software quality metrics methodology. Technical report, IEEE.

[IEEE, 2011] IEEE (2011). ISO/IEC/IEEE 29148 - Systems and software engineering — Life cycle processes — Requirements engineering. Technical report, IEEE.

[IEEE, 2017] IEEE (2017). ISO/IEC/IEEE 15939:2017(E) - Systems and software engineering - Measurement Process. Technical report, ISO/IEC/IEEE.

[Johnson, 2007] Johnson, P. M. (2007). Requirement and design tradeoffs in Hackystat: An inprocess software engineering measurement and analysis system. In *1st International Symposium on Empirical Software Engineering and Measurement*, pages 81–90, Madrid, Spain. IEEE.

[Kerzner, 2017] Kerzner, H. (2017). *Project management metrics, KPIs, and dashboards: A guide to measuring and monitoring project performance.* John and Wiley & Sons, third edition.

[Mendonça and Basili, 2000] Mendonça, M. G. and Basili, V. R. (2000). Validation of an approach for improving existing measurement frameworks. *IEEE Transactions on Software Engineering*, 26(6):484–499.

[Offen and Jeffery, 1997] Offen, R. J. and Jeffery, R. (1997). Establishing software measurement programs. *IEEE Software*, 14(2):45–53.

[Ohira et al., 2004] Ohira, M., Yokomori, R., and Sakai, M. (2004). Empirical project monitor: A tool for mining multiple project data. In *Proceedings of the International Workshop on Mining Software Repositories (MSR'04)*, pages 42–46, Edinburgh, UK.

[Olsson and Runeson, 2001] Olsson, T. and Runeson, P. (2001). V-GQM: A feed-back approach to validation of a GQM study. In *Proceedings of the 7th International Software Metrics Symposium*, pages 236–245. IEEE.

[Pauwels et al., 2009] Pauwels, K., Ambler, T., Clark, B. H., LaPointe, P., Reibstein, D., Skiera, B., Wierenga, B., and Wiesel, T. (2009). Dashboards as a service. *Journal of Service Research*, 12(2):175–189.

[Rowe and Whitty, 1993] Rowe, A. and Whitty, R. (1993). Ami: promoting a quantitative approach to software management. *Software Quality Journal*, 2:291–296.

[Selby, 2009] Selby, R. W. (2009). Analytics-driven dashboards enable leading indicators for requirements and designs of large-scale systems. *IEEE Software*, 26(1):41–49.

[Shaw, 2003] Shaw, M. (2003). Writing good software engineering research papers. In *25th International Conference on Software Engineering (ICSE 2003)*, pages 726–736, Portland, Oregon. IEEE.

[Sillitti et al., 2003] Sillitti, A., Janes, A., Succi, G., and Vernazza, T. (2003). Collecting, integrating and analyzing software metrics and personal software process data. In *Proceedings of the 29th EUROMICRO Conference*, pages 336–342, Belek-Antalya, Turkey. IEEE Computer Society.

[Susman and Evered, 1978] Susman, G. and Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative Science Quarterly*, 23(4):582–603.

[Tennant, 2001] Tennant, G. (2001). *Six Sigma: SPC and TQM in manufacturing and services*. Gower, Burlington, USA.

[Wiegers, 2006] Wiegers, K. E. (2006). *More about software requirements: Thorny issues and practical advice*. Microsoft Press, Washington.

[Zhong et al., 2004] Zhong, S., Khoshgoftaar, T. M., and Seliya, N. (2004). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems*, (March/April).

Chapter 5

A Health Assessment Framework for Systems Projects: A Requirements Perspective

5.1 Introduction

Significant effort has been made in the software engineering (SE) community to understand the critical success factors (CSF) of software and system development projects, which includes, among others, organizational (e.g., management support, organizational culture, and leadership), team-related (e.g., communication, expertise, and composition), and technical (e.g., software development methodology, supporting tools, and requirements volatility) factors that have significant impact on project success [Ahimbisibwe et al., 2015; Mohd and Shamsul, 2011; Chow and Cao, 2008]. Particularly, evidence shows that requirements attributes such as growth [Boehm, 1991; Jones, 2000], volatility [Zowghi and Nurmuliani, 2002; Ferreira et al., 2009], and requirements defects [Boehm and Basili, 2001; Jones and Bonsignour, 2012] have a significant impact on project time, cost, and quality and, thus, overall project success.

While project success is the final verdict on a project *after* completion that indicates it was on time, within budget, and delivers value to the customer [Jones, 1996a], project *health* indicates its likelihood of success *during* a project. Thus, project health assessment is a complex task of understanding, assessing and monitoring numerous critical factors that are likely to impact the project's likelihood of success. Health checks can be performed throughout a software or system project's life cycle stages. The benefits of health checks include: proactively identifying problems so as to allow sufficient time for taking corrective action, identifying which development activities or areas of the project that may require additional resources, identifying present and future risks and the possible risk mitigation strategies, and developing recommendations for a fix-it plan [Kerzner, 2017a,b].

However, there is a gap between the CSF literature and assessing project health in practice, which is usually limited to the monitoring of project factors (e.g., cost, quality, and time) as indicators of project health [Kerzner, 2017a,b; Bittner, 2007]. Particularly, there are no systemic, holistic health assessment approaches or frameworks that incorporate requirements with project data to provide a requirements-centric view of project health despite the critical role that requirements play in determining a project's likelihood of success.

Our experience with systems projects shows that the need for such frameworks becomes even more critical within the context of large system projects that have numerous subprojects, each of which has it own large set of requirements and software artifacts (i.e., design objects, tests, and code), schedules, and deadlines, to name a few. Thus, assessing and monitoring project health is of utmost importance in order to ensure that the project is heading towards successful completion and delivery. Particularly, there is a need to track project health from a requirements perspective in order to identify critical requirements-related problems as they manifest and take action to resolve them to avoid project failure. However, to our knowledge, there are no frameworks to assist in this essential task and health assessments are performed in an ad-hoc manner by a select few who have an understanding of the many facets of the projects.

Thus, our research objective is to provide a measurement-based framework that enables the identification of project health indicators based on a holistic assessment of requirements measures in conjunction with relevant project data (e.g., deadlines and resources) to provide a high-level view of project health from a requirements perspective with the ability to drill down to specific, raw measures. The purpose is to aid manual and personal-based assessments with a quantitative and systematic assessment of project health that can aid in identifying critical requirements-related problems that may negatively impact the project's likelihood of success.

To achieve this objective, this paper presents four main contributions: 1) A conceptual architecture of a measurement-based framework that incorporates established measurement concepts with CSFs and project data to derive project health indicators (Section 5.4.1). 2) An operationalization of the framework within the context of system projects based on empirical and literature-based understanding of requirements, measurement and project health (Section 5.4.2). 3) Tool support of the operationalized framework, which is implemented as a feature in a requirements dashboard (Section 5.5). 4) Preliminary evaluation of the framework by implementing it on real data from three systems projects in the rail automation domain and comparing the framework's results with manual assessments of one of the projects (Section 5.6).

5.2 Related Work

In this section, we discuss the related work on project health assessment from the project management, software engineering (SE) and requirements engineering (RE) literature and subsequently discuss the research gap.

The body of work in the project management literature on project health assessment is larger than its counterpart in the SE literature and consists of approaches, tools, and checklists that enable interested parties to gauge project health at specific points in time during the project. At a high level of abstraction, Snider et al. [Snider et al., 2018] present a framework, Engineering Project Health Management (EPHM), that adapts the integrated vehicle health management (IVHM) framework from the aeronautic industry to the field of engineering management. EPHM monitors engineering work (e.g., design, modeling, and communication) through data-driven and computational analytics that, in turn, support the generation of higher level, context-specific knowledge. The framework is a high-level abstraction of the general activities needed to elicit project health: data acquisition, data manipulation, state detection, and presentation. The details of the activities and data within each level is context dependent and, thus, not delineated in that paper, nor project attributes that aid in assessing project health. However, the authors define three levels of attributes (physical, content, and context) that can be measured for three types of assets in an engineering environment: communication (e.g., emails and instant messaging), representation (e.g., prototypes and analysis models), and project report/record (e.g., technical reports and databases). They recommend the use of statistical means to compare current project states with past states, historical cases, and trends with time and suggest using the traffic lights system (red, amber, green) to visualize the data and provide quick understanding of areas of interest.

On a more operational level, Jaafari's [Jaafari, 2007] health assessment approach, referred to as the project health check (PH-Check), assesses the practices applied to manage a set of project variables such as risk management, change management, internal efficiency, quality assurance, and others. Each variable is assessed manually and given a percentage. The assessment is based on a comparison between the current state of practice and a given, five-level state of practice. Similarly, Philbin and Kennedy [Philbin and Kennedy, 2014] present a health diagnostic framework that includes review criteria for the process, technology, resources, impact, and knowledge dimensions of a project. The assessment is to be done manually by using the review criteria. While not an approach per se, Bittner [Bittner, 2007] argues that different attributes must be measured at different points in the project in order to assess project health. The project can be divided into four phases: inception, elaboration, construction, and transition. He focuses on the inception phase and discusses in depth the assessment of financial viability, technical viability, and project viability as the core project attributes that must be measured to assess project health during the inception phase. He briefly discusses other measures that can be used as indicators for project health: customer satisfaction, morale, absolute progress, risk stability, and requirements stability.

The previous work focused on measuring project attributes as indicators of project health in general. In SE, some work has been done on measuring software and system attributes (e.g., defect density, and code churn) to assess project health, which have been shown to be associated with project success and failure [Jones and Bonsignour, 2012]. For instance, Piggot and Amrit [Piggot and Amrit, 2013] use machine learning decision trees to create a model that determines the stage (i.e., planning, pre-alpha, alpha, mature) of an open source software project through the use of eight metrics: number of bugs, number of downloads, number of forum posts, number of developers, number of donors, number of commits, number of service requests, and operating system type. Sharma and Kaulgud [Sharma and Kaulgud, 2012] present PIVoT (Project Insights and Visualization Toolkit), a tool that provides project managers with a holistic picture of a project's health and trajectory. PIVoT collects and visualizes thirty software metrics related to code quality, quality of component testing effort, development efficiency, code churn, and team analysis. The measures are calculated and visualized in the tool through an assortment of graphs. Finally, the Q-Rapids method and tool [Franch et al., 2018] is part of large project that gathers data from different software and project repositories (e.g., Jira and SonarQube), analyzes the gathered data, and presents it at three levels of abstraction: metrics (e.g., code bug density and availability uptime), product/process factors (e.g., code quality and software usage), and strategic indicators (e.g., customer satisfaction and product quality).

The above review of the literature demonstrates that the notion of using requirements attributes as indicators of project health is, to our knowledge, non-existent in the literature. We commonly find that requirements attributes are studied in isolation. For example, many studies have been conducted to understand the effect of requirements volatility on project success and failure [Pena and Valerdi, 2014; Ferreira et al., 2011b; Malaiya and Denton, 1999]. We also find studies that attempt to identify thresholds for such requirements measures (e.g., volatility [Jones, 1996a] and creep [Kulk and Verhoef, 2008; Jones, 1996b]). Moreover, the idea of measurement frameworks that measure certain requirements attributes to create 'indices' have largely focused on requirements quality attributes (e.g., size, complexity, and ambiguity). For example, Genova et al. [Génova et al., 2013] select a number of desirable requirement quality attributes (e.g., completeness, consistency, and abstraction), define means to measure said attributes, and provide methods to create indices that provides an overall idea of the quality of the textual requirements. Similarly, Antinyan and Staron [Antinyan and Staron, 2017] define requirements size, complexity, and coupling metrics, which are then used in a formula to extract a quality index. While the high-level aim of Q-Rapids [Franch et al., 2018] echoes our aim in attempting to analyze large amounts of data and providing indicators of a sort to the internal stakeholders, our work differs and complements Q-Rapids by focusing on requirements metrics and eliciting requirements-centric health indicators as opposed to development-level metrics and indicators. To our knowledge, a holistic framework that assesses a combination of requirements attributes and analyzes their measures based on historical data and in conjunction with project attributes to assess project health has not been studied. This study attempts to fill this gap.

5.3 Context Description

As discussed in the introduction, the motivating factors that led to the creation of the framework emerged from a large systems project. To better understand that context and, in the following sections, connect the usage of the proposed framework to its originating environment, we briefly describe the organizational context here.

The overall program consisted of numerous projects in a large, safety critical, systems engineering domain, three of which we were directly involved with. Each project consisted of a product that had its own set of requirements, architecture design, test cases, contracts, deadlines, and engineering team (e.g., requirements managers, architects, RD managers, and quality and safety managers). Each project had of a large number of requirements, design artifacts, and tests. Table 5.1 shows a breakdown of the software artifacts for three projects, which consists of the approximate number of requirements baselines (i.e., document versions), requirements, design baselines, design objects, and test cases per product and that are continuously increasing over the lifetime of the project.

On the management level, a person is assigned to manage a subset of projects, particularly with regard to the project's requirements and their progress in relation to downstream processes. The person is expected to gauge whether the project is doing well in terms of progress and other factors (deadlines, quality, and coverage, etc.) and to identify problematic areas that need to be addressed to avoid project failure [Jones, 1996a]. This is done manually depending largely on a select few's expertise who have knowledge of a wide range of project and software factors. There is no quantitative, systematized approach to assess project health from a requirements perspective. Thus, project and program-level monitoring of requirements and preemptive identification of potential problems becomes critical.

Table 5.1: Descriptive statistics of context.

Project	**Project Duration**	**No. Req. Baselines**	**No. Reqs.**	**No. Safety Reqs.**	**No. Design Baselines**	**No. Design Objects**	**No. Test Cases**
P1	73 months	54	1790	N/A	23	472	2111
P2	36 months	30	2285	N/A	4	380	N/A
P3	45 months	51	2389	923	28	827	2045

5.4 The Health Assessment Framework

In this section, we first describe the conceptual architecture of the framework that defines the framework's components and how they relate to each other. Then we discuss the operationalization of the conceptual architecture based on data from the literature and three systems projects in the rail-automation domain.

5.4.1 Conceptual Architecture of the Framework

With reference to Figure 5.1, the project health assessment framework is a measurement-based framework that consists of a number of measurement-related components adapted from [IEEE, 1992] and [Staron and Meding, 2018] and project health components from the project health and CSF literature. At its essence, the framework relies on the selection of measurable *attributes* (e.g., status, complexity, and volatility) of our software entity of interest (i.e., requirements) that ultimately impact project health in a positive or negative way. To make the connection between the measurable attribute and the more abstract notion of project health, we added the concept of *critical success factors* (CSF) to the framework. CSFs are issues that, if addressed appropriately, substantially increase the likelihood of project success [Ahimbisibwe et al., 2015]. Thus, project health assessment depends on measuring and monitoring attributes that are dimensions of related critical success factors. For example, user/client participation is considered as one of the top three CSFs for agile and traditional plan-based software projects [Ahimbisibwe et al., 2015]. Thus, a project health assessment check may consist of measuring frequency of communication between client and vendor and amount of user/client feedback, both of which are dimensional attributes of the user/client participation CSF. The identification of requirements attributes that are dimensional to CSFs allows for the explicit consideration of project health when measuring requirements.

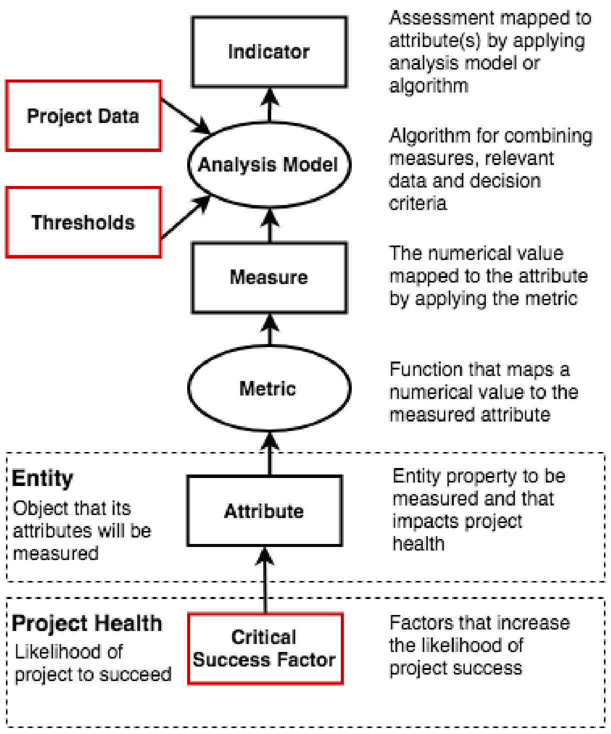

Indicator
Assessment mapped to attribute(s) by applying analysis model or algorithm
Project Data
Analysis Model
Algorithm for combining measures, relevant data and decision criteria
Thresholds
Measure
The numerical value mapped to the attribute by applying the metric
Metric
Function that maps a numerical value to the measured attribute
Entity
Object that its attributes will be measured
Attribute
Entity property to be measured and that impacts project health
Project Health
Likelihood of project to succeed
Critical Success Factor
Factors that increase the likelihood of project success

Figure 5.1: The conceptual architecture of the framework.
Depicted in this figure are the main conceptual components of the health assessment framework. The black boxes are measurement-related components based on established measurement concepts. The red boxes are novel additions in comparison to existing measurement frameworks in RE. The dotted boxes represent abstract concepts that relate to and inform the framework but are not considered technical parts of it.

The selected attributes are then quantified using a *metric*, which is a "function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possess a given attribute." [IEEE, 1992] While the general understanding of an attribute is based on scientific evidence from the metrics literature (e,g., requirements volatility), the metric definitions (i.e., functions) are based on the organization's understanding of how a specific attribute should be quantified [Wagner et al., 2012].The numerical value assigned to an attribute as the result of applying the metric is called a *measure*. For instance, the LOC metric is "the number of lines of code in a software" and the LOC measure for a software system is, say, 2000. The metrics and measures provide a quantitative basis for project health assessment as opposed to a manual, assumption-based assessment.

While sole measures of requirements attributes related to project health is an improvement over the complete absence of measures in assessing project health, sole requirements measures do not provide a holistic assessment of project health. Measures must be assessed in relation to thresholds to determine what is an acceptable value for an attribute. Moreover, external (i.e., non-software) project factors (e.g., schedule, cost, and resources) play a significant role in determining the criticality of certain measures [Kulk and Verhoef, 2008]. Thus, the explicit identification of project data and thresholds is, to our knowledge, new in requirements measurement, as measures are usually assessed in isolation.

The measures, thresholds, and relevant project data are then analyzed using an *analysis model*, which consists of a set of criteria and conditions that reflect an organization's understanding of project health in relation to the inputs (i.e., measures, thresholds, and project data). The result of the analysis model is an *indicator* for the measured attribute, which can be represented in numerous forms; numeric, textual, or graphic, to name a few.

5.4.2 Operationalization of the Framework

In this subsection we discuss the operationalization of the above concepts, which was based on: i) the empirical input of the practitioners' expertise and experience in the collaborating organization and ii) scientific support from the literature. We discuss the operationalized CSFs, requirements attributes, metrics, thresholds, project data, analysis models, and indicators in detail below.

CSFs, Requirements Attributes, and Metrics

Table 5.2 shows the requirements attributes that were selected for measurement, the CSF they relate to, the metrics we defined to quantify the selected attributes, and the impact the attribute has on project health. A negative impact means the higher the measure of a specific attribute the more negative impact it has on project health and a positive impact means the higher the measure the more healthy the project. It is important to note, however, that the selection of attributes is not by any means exhaustive; we were constrained by the availability of data (see below for a discussion of requirements attributes that were considered but not used). We discuss each attribute in more detail in the following paragraphs.

Requirements Growth. Growth, also referred to as requirements creep, is the percentage of increase or decrease in requirements size between an older and newer requirements baseline [Jones, 2000]. Empirical evidence suggests that unsuccessful projects have more than a 30% growth/creep rate in user requirements while successful ones have less than 10% requirements growth [Boehm, 1991]. Specifically, requirements creep over 5% in systems software is considered a failure factor [Jones, 2000]. Other sources suggest a monthly 1.25% monthly rate for applications of the size of 1000 functions points [Jones and Bonsignour, 2012]. Moreover, requirements changes or introducing new requirements increase the de-

Table 5.2: Operationalized CSFs, attributes, and metrics.

Related CSF	**Requirements Attribute**	**Metric ID**	**Metric**	**Impact on Project Health**
Requirements stability [Jones, 2000; Ahimbisibwe et al., 2015; Ferreira et al., 2011b]	Growth	RG	(difference between the number of requirements of the first baseline and the latest baseline) / (number of the old baseline)	Negative
Requirements stability [Jones, 2000; Ahimbisibwe et al., 2015; Ferreira et al., 2011b]	Volatility	RV	(number of deleted, added, and modified requirements in the latest baseline in relation to the preceding baseline) / (number of requirements in the latest baseline)	Negative
Up-to-date progress reporting and effective monitoring and control [Mohd and Shamsul, 2011]	Requirements-Design Coverage	RDC	(number of requirements with links to design objects) / (number of requirements in the latest baseline)	Positive
Up-to-date progress reporting and effective monitoring [Mohd and Shamsul, 2011] and quality assurance [Niazi et al., 2006]	Requirements-Test Coverage	RTC	(number of requirements with links to tests) / (number of requirements in the latest baseline)	Positive
Quality assurance [Niazi et al., 2006]	Requirement Defect Removal Efficiency	DRE	(number of closed requirements defects) / (total number of requirements defects)	Positive

fect rate to about 50% [Park et al., 2010]. It comes as no surprise, then, that requirements stability is considered a CSF [Ahimbisibwe et al., 2015; Jones and Bonsignour, 2012; Ferreira et al., 2011b]. Thus, requirements growth is a good indicator of project health. Specifically, lower requirements growth rates indicate better project health and, thus, increased likelihood of project success.

Requirements Volatility. Volatility is the extent to which requirements change in response to factors such as the changing needs of customers, stakeholders, and organization, innovation, market forces, regulations, and others [Ferreira et al., 2009]. This manifests as additions, deletions and modifications to project requirements over time [Costello and Liu, 1995]. Evidence shows that an increase in requirements volatility decreases the probability of a project to complete on time as well as on budget [Zowghi and Nurmuliani, 2002]. More specifically, increases in the project job size and rework lead to increases in the amount of effort required to complete the project. These increases in effort cause the cost of the project to rise if additional resources need to be added to accommodate the extra workload. If the schedule is not changed and/or no more resources are added to address the additional workload, schedule pressure also rises [Ferreira et al., 2011b]. Thus, requirements volatility is a good indicator of project health and must be monitored and managed accordingly. Specifically, lower requirements volatility means a healthier project.

Requirements-Design Coverage. Requirements-design coverage is the degree to which a set of requirements are covered by a software design. In other words, it is the percentage of requirements that have identifiable links to software design. Studies show that requirements-design coverage metrics can be used to monitor the status of the requirements during development as they progress from planned to implemented and tested, assess the development process, detect inconsistencies and incompletenesses, and assess defect and change rates [Winkler and von Pilgrim, 2010]. Additionally, coverage information between requirements and design was chosen as a core traceability link that needs to be reported to developers and managers for release planning [Lago et al., 2009]. These factors directly impact project cost and schedule. Thus, requirements-design coverage is indicative of project health; higher requirements-design coverage rates mean further progress within the project and fewer inconsistencies and incompletenesses between requirements and design, which, in turn leads to lower defect and change rates.

Requirements-Test Coverage. Requirements-test coverage is the degree to which a set of requirements have been covered by tests. Higher coverage between requirements and tests leads to fewer defects [Damian and Chisan, 2006] (i.e., quality) and allows the identification of "excessive or insufficient test cases, and incomplete linkage of tests to requirements" [Rosenberg et al., 1998] (i.e., progress). Therefore, requirements-test coverage is a good indicator of project health.

Requirements Defect Removal Efficiency. It is the percentage of requirements defects resolved in relation to the total number of requirements defects per requirements baseline. Literature suggests that requirements-related defects are a very costly problem to fix [Park et al., 2010]. While some studies suggest that the cost of fixing requirements defects may rise by 20 to 50 times if the defects are fixed in the later stage of the development [Fairley, 1985], others put that number as high as 100 times [Boehm and Basili, 2001]. Moreover, evidence suggests that requirements defects generate 20% of the total volume of software defects [Jones and Bonsignour, 2012]and other raise that number up to 85% of software defects [Hooks and Farry, 2001]. This means that the higher the requirements defects removal efficiency indicates better quality assurance and, thus, better project health.

Attributes considered but not measured. We considered including requirements size or product size (i.e., number of specified and implemented project requirements) as an indicator of project health. However, by itself, requirements size does not give any insight into project health; it must be assessed in relation with other project factors, such as project duration and team size. For example, if the optimal team size and project duration for a product with 2000 requirements is 50 team members and 36 months, respectively, then a product with 2000 requirements and 35 team members and a duration of 24 months may indicate low project health. We did not include requirements size in our framework due to lack of appropriate historical data for comparison purposes.

Requirements quality is another requirements attribute that could potentially enrich the informational value provided by the health assessment framework. For instance, a requirements quality health indicator derived from requirement quality measures (e.g., complexity, precision, and size) [Antinyan and Staron, 2017; Génova et al., 2013] and project data could provide insight into project health from a requirements perspective. However, the quality aspect is not included in the operationalized framework at this stage of research.

The Case of Quality Requirements. Given the focus on requirements in our framework, we briefly discuss our treatment of requirements as a measurable entity within the context of our proposed framework. As discussed in the previous section, the identified attributes and associated metrics apply to the entire set of requirements for a given product. However, depending on the application domain, certain quality attributes become relatively more important. For example, in safety critical systems, safety requirements are clearly of utmost importance and the project's health from this perspective must be monitored as appropriate. Thus, the above attributes and metrics apply to the entire set of requirements and the subset of safety requirements. In other words, we would measure growth, volatility, design coverage, test coverage, and defect efficiency removal for *all* project requirements and similarly for *safety* requirements. The selection of quality requirements that need to be assessed is context dependent.

Thresholds

A solitary measure does not tell us anything about the attribute it measures without a reference value. For example, a requirements volatility measure of 30% is valuable only in relation to established norms in practice and literature that indicate that requirements volatility should not exceed a rate of 5% [Jones, 2000]. While the literature provides some benchmark rates for requirements growth [Kulk and Verhoef, 2008] and volatility [Jones, 2000], acceptable rates of measures remain highly context dependent. Thus, we use two threshold identification methods from the literature that allow the definition of thresholds based on the used set of data. We use Ferreira's et al. method [Ferreira et al., 2011a] for growth and volatility thresholds and Genova's et al. method [Génova et al., 2013] for design coverage, test coverage, and defect removal efficiency thresholds.

Ferreira's method [Ferreira et al., 2011a] consists of identifying three ranges for a measure: good, which refers to the most common values of the metric; regular, which is an intermediate range of values with low frequency, but not irrelevant; and bad, which refers to values with rare occurrences. This entails the calculation of the frequency values for each measure then identifying the three ranges based on those frequencies. For example, we may find that requirements growth is less than 2% per baseline for 70% of the baselines (good requirements growth), between 2% and 10% growth for 20% of the baselines (regular requirements growth), and more than 10% growth for 10% of the baselines (bad requirements growth). Thus, reasonable ranges for requirements growth would be (0-2, 2-10, 10+). These thresholds do not necessarily denote the best practices in software engineering, but represent the levels of the specific project and what is it used to handling [Ferreira et al., 2011a]. The same method is used for requirements volatility. This method rests on the premise that the values of the measures follow a power law [Ferreira et al., 2011a]. Meaning, that the frequency of high values for the metrics is very low, while frequency of low values is high, which is characteristic of requirements growth and volatility. This is tested by plotting the data in a histogram and fitting the data to a probability distribution as depicted in Fig. 5.2.

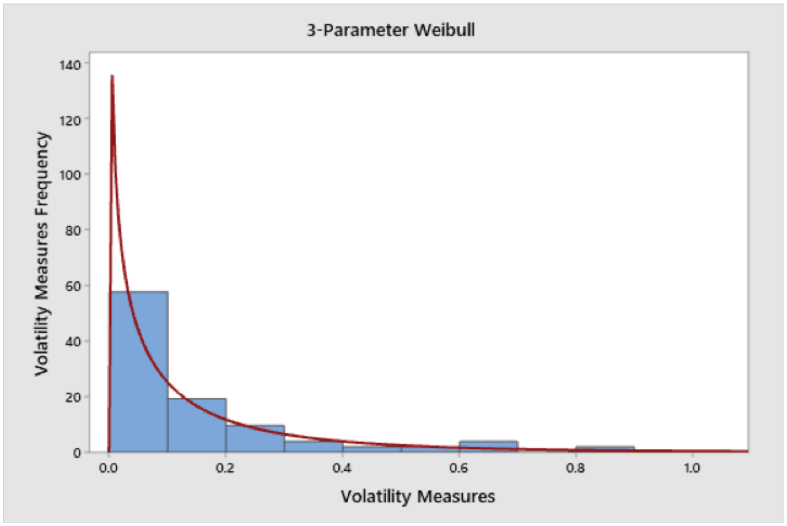


Figure 5.2: Distribution and histogram of volatility measures.
This figure depicts the distribution of volatility measures for a given project over 51 requirements baselines. We see that close to 80% of the measures are <0.2 and that the data fits a a 3-parameter Weibull distribution, which does not have a representative mean value. Thus, we choose the good, regular, and bad threshold ranges.

We use Genova's et al. [Génova et al., 2013] threshold identification method for design coverage, test coverage, and defect removal efficiency because—unlike volatility and growth measures—coverage and defect removal efficiency values do not follow a power law but begin at 0% and are expected to be 100% at the end of the project or by a specific deadline. The method utilizes increasing step functions to identify coverage and defect removal efficiency thresholds, which consists of classifying measures into a set of discrete levels where numbers start from 0 and larger. Table 5.3 shows the ranges identified for each requirement attribute based on the above threshold methods.

Table 5.3: Threshold and project data operationalization for each requirement attribute.

Attribute	**Threshold Identification Method**	**Threshold Ranges**	**Project Time Ranges**	**Number of Conditions**	**Applicable to Quality Requirements?**
Growth	Ferreira's method [Ferreira et al., 2011a]	Good, regular, bad	Early, mid, late	9	Yes
Volatility	Ferreira's method [Ferreira et al., 2011a]	Good, regular, bad	Early, mid, late	9	Yes
Design Coverage	Genova's method [Génova et al., 2013]	Very low, low, low medium, high medium, high, full	Early, early mid, late-mid, late	24	Yes
Test Coverage	Genova's method [Génova et al., 2013]	Very low, low, low medium, high medium, high, full	Early, early mid, late-mid, late	24	Yes
Defect Removal Efficiency	Genova's method [Génova et al., 2013]	Very low, low, low medium, high medium, high	Early, early mid, late-mid, late	20	Yes

Project Factors

The proposed framework takes two external project factors into account when assessing project health: project time and deadlines.

Project Time. There is considerable evidence that shows that growth and volatility rates tend to be higher at the beginning of the project than at the end [Kulk and Verhoef, 2008]. Thus, a specific measure may be considered acceptable at the beginning of a project but problematic at the end of the project. Similarly, requirements-design coverage, requirements-test coverage, and defect removal efficiency naturally begin at 0 at the beginning of the project and gradually increase over the life of a project. Therefore, while the identified threshold ranges may deem a 5% requirement-design coverage as low, it is not logical to raise alerts when the project is in its early phase. Thus, we identified project time ranges (early, early mid, late mid, and late) that the measures, while taking their threshold range into consideration, must be analyzed against. Table 5.3 shows the project time ranges identified for each requirement attribute.

Multiples Deadlines. The project time ranges assume that a project has a start date and an end date, or, in other words, one deadline. However, in reality, large projects have multiple, interim deadlines within the entire duration of the project. If a project has multiple deadlines, then project health assessment at a specific point in time must consider the closest deadline to the current date, and identify the project health in relation to the project time range that the upcoming deadline falls into. For example, if the current time of the project is in the early stages of the project but there is an upcoming deadline that falls in the mid stage of a project. Thus, the requirements measures must be assessed in relation to the upcoming deadline.

We note that project time and deadlines are not the only project factors that impact the health of the requirements attributes taken into consideration in the framework. For instance, project resources are another factor that may impact the health of the above attributes. Abundant resources (e.g., man power and effective technologies) may allow for more effective management and control of higher volatility rates for instance, which, in turn, may indicate better project health. However, we restrict our selection to project time and deadlines as they were the most accessible data to measure.

Analysis Model

As depicted in Figure 5.1, the analysis model of the framework analyzes the calculated requirements measures, thresholds, and project data to derive the health indicators. Each requirements attribute has its corresponding analysis model and consists of four general phases that we describe below. Figure 5.3 depicts the flowchart for the requirements *growth* analysis model, which we will use as an illustrative example.

1. Retrieve Relevant Data. The green steps depicted in Figure 5.3 retrieve current growth measure, its threshold range, current project time, number of project deadlines, and project time ranges to calculate the health indicator for each requirements attribute.

2. Analyze Project Data. The orange steps in Figure 5.3 depict the analysis of the project data. In a nutshell, we first test whether the project has more than one deadline. If yes, the analysis model goes into the next phase. If there's more than one deadline, then depending on which project range the upcoming deadline is in, we set the current project time accordingly. For example, as depicted in Figure 5.3, if the current project time is in the early project range and the upcoming deadline is in the mid project range, then the current growth measure should be treated as it is in the mid project range. Thus, we assign the current project time to mid.

3. Analyze Requirement Measures and Project Data. The pink steps in Figure 5.3 show the series of conditional statements that the analysis model tests. Based on current requirements measures, identified thresholds, and project data, a red, amber, or green indicator is assigned to the attribute. We discuss the meaning of each color in Section 5.4.2 below. The assignment of indicators to conditions were identified to reflect the organization's development culture and practices.

4. Reapply Steps for Quality Requirements. To identify the growth health indicator for

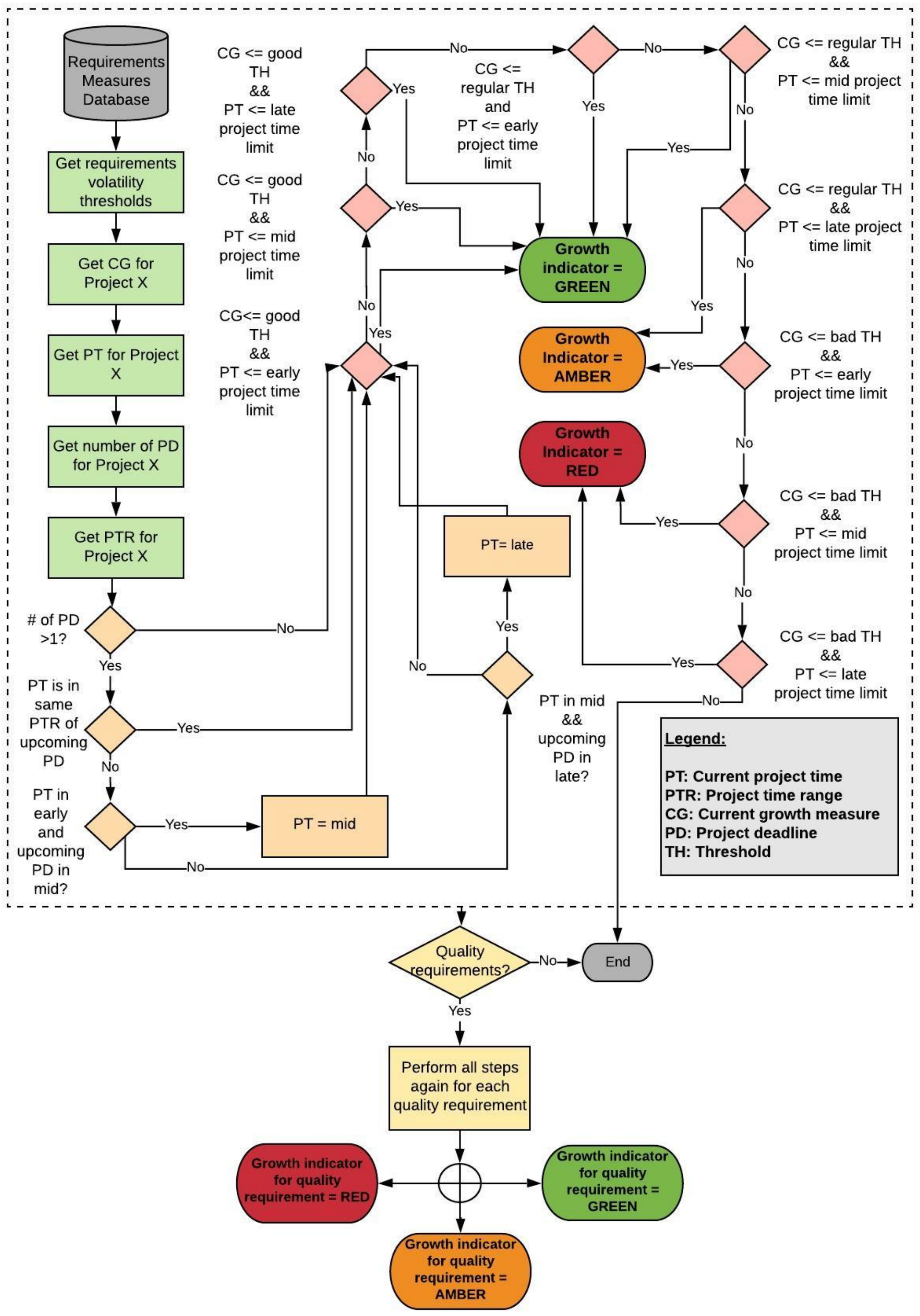

Requirements Measures Database
Get requirements volatility thresholds
Get CG for Project X
Get PT for Project X
Get number of PD for Project X
Get PTR for Project X
# of PD >1?
PT is in same PTR of upcoming PD
PT in early and upcoming PD in mid?
PT = mid
PT= late
PT in mid && upcoming PD in late?
CG <= good TH && PT <= late project time limit
CG <= good TH && PT <= mid project time limit
CG<= good TH && PT <= early project time limit
CG <= regular TH and PT <= early project time limit
CG <= regular TH && PT <= mid project time limit
CG <= regular TH && PT <= late project time limit
CG <= bad TH && PT <= early project time limit
CG <= bad TH && PT <= mid project time limit
CG <= bad TH && PT <= late project time limit
Growth indicator = GREEN
Growth Indicator = AMBER
Growth Indicator = RED
Yes
No
Legend:
PT: Current project time
PTR: Project time range
CG: Current growth measure
PD: Project deadline
TH: Threshold
Quality requirements?
End
Perform all steps again for each quality requirement
Growth indicator for quality requirement = RED
Growth indicator for quality requirement = GREEN
Growth indicator for quality requirement = AMBER

Figure 5.3: Requirements growth analysis model.

quality requirements, the previous steps are reapplied to the quality requirement of interest (yellow steps in Figure 5.3). For example, current requirements growth measure for safety requirements will be retrieved from the requirements measures database and the analysis model steps are executed to identify its corresponding health indicator.

Each requirement attribute has a similar analysis model with variations in conditions. Table 5.3 summarizes the operationalization of the thresholds ranges, project ranges and the number of conditions in the analysis model for each requirement attribute.

Project Health Indicators

Figure 5.4 shows an example of the result of executing the analysis models for each requirements attribute. We use the red-amber-green (RAG) indicator system [Staron and Meding, 2018] to visualize the indicators. An indicator for each attribute for *all* project requirements and another five health indicators for *quality* requirements considered in the framework as depicted in Figure 5.4b.

A *green* indicator means no action is required; the requirement attribute is healthy and no concerns are raised with regard to its measure, the current project time and any upcoming deadlines. An *amber* indicator means some sort of action is recommended to avoid severe problems. For example, bad volatility rates in the mid phase of the project requires an investigation into the reason behind the high volatility rate and taking action to mitigate its consequences. A *red* indicator means immediate action is needed. For example, the project is in its late phase with a very low level of defect resolution.

The individual indicators depict the health of each attribute separately. To get on overall picture of *project* health, the five indicators are aggregated into one project health indicator and displayed as a pie chart that shows the ratios of the colors as depicted in Figure 5.4a. Currently, all attributes are weighted equally and, thus, each attribute equals twenty percent. For example, a project indicator may show a project to be 60% green, 20% amber, and 20% red. Because the project health indicator is the first and highest-level indicator to be displayed to the internal stakeholders, the stakeholder has the ability to drill down to investigate the constituents of each indicator (i.e., growth, volatility, and coverage, etc.) and then further down to the raw measures for each attribute.

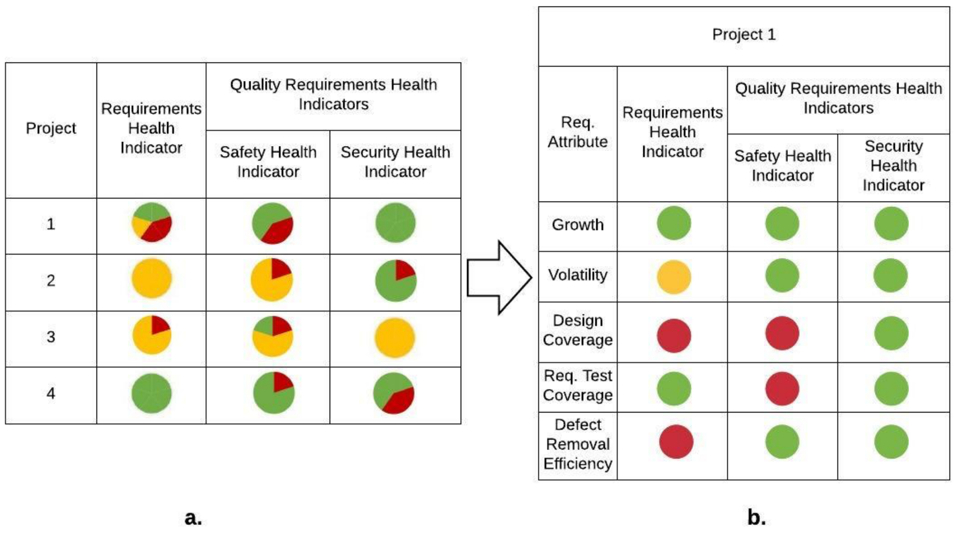


Figure 5.4: Requirements-centric health indicators visualized using the RAG system.

5.5 Tool Support

We implemented and integrated the health assessment framework as part of a requirements measurement dashboard that aims to provide internal stakeholders with a large number of requirements measures (e.g., status, volatility, and growth) and associated visualizations. While a detailed description of the dashboard is not included in this paper, we describe the requirements health indicators feature of the dashboard that implements the framework.

Figure 5.5 depicts the dashboard which is composed of a 3-tier architecture and consists of a database layer, application layer, and presentation layer. The application layer is implemented in Java and consists of packages, each of which handles different features of the dashboard. The health assessment framework is handled by the *middleware.healthindicators* package and implements the analysis models described in Section 5.4.2. It retrieves the required data from the requirements measures database that contains projects, requirements baselines, requirements attributes, associated metrics and measures, and users, among others, through a *MetricService* class. The package also uses the R language to perform the statistical analyses required for the thresholds as described in Section 5.4.2.

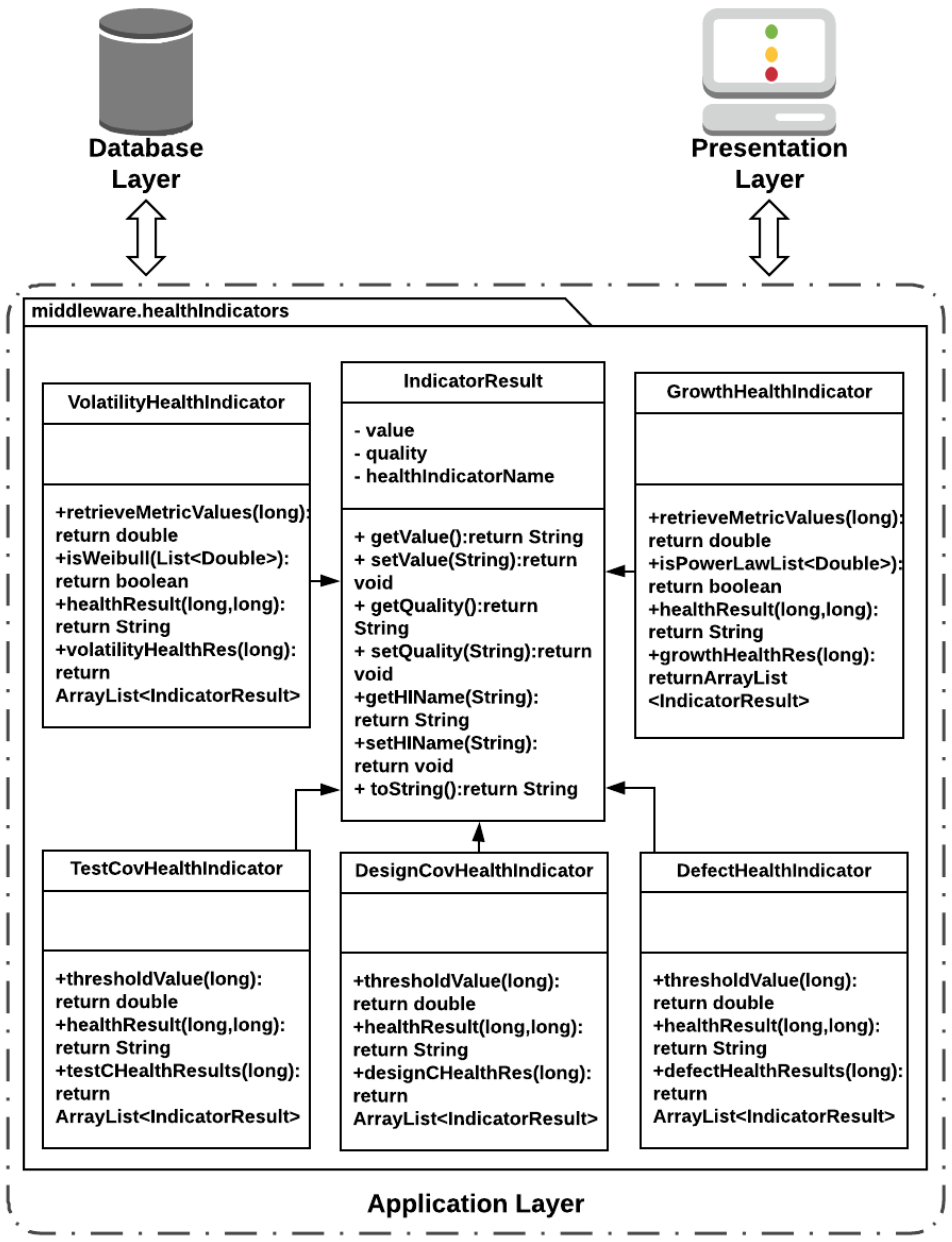

Database Layer
Presentation Layer
middleware.healthIndicators
VolatilityHealthIndicator
+retrieveMetricValues(long): return double
+isWeibull(List<Double>): return boolean
+healthResult(long,long): return String
+volatilityHealthRes(long): return ArrayList<IndicatorResult>
IndicatorResult
- value
- quality
- healthIndicatorName
+ getValue():return String
+ setValue(String):return void
+ getQuality():return String
+ setQuality(String):return void
+getHIName(String): return String
+setHIName(String): return void
+ toString():return String
GrowthHealthIndicator
+retrieveMetricValues(long): return double
+isPowerLawList<Double>): return boolean
+healthResult(long,long): return String
+growthHealthRes(long): returnArrayList <IndicatorResult>
TestCovHealthIndicator
+thresholdValue(long): return double
+healthResult(long,long): return String
+testCHealthResults(long): return ArrayList<IndicatorResult>
DesignCovHealthIndicator
+thresholdValue(long): return double
+healthResult(long,long): return String
+designCHealthRes(long): return ArrayList<IndicatorResult>
DefectHealthIndicator
+thresholdValue(long): return double
+healthResult(long,long): return String
+defectHealthResults(long): return ArrayList<IndicatorResult>
Application Layer

Figure 5.5: Architecture of the requirements dashboard.
This figure depicts the 3-tier (database, application, and presentation) architecture of the requirements dashboard that includes the health assessment feature, implemented in the application layer through the *middleware.healthindicators* package and its results presented through the presentation layer.

5.6 Evaluation

The objective of the evaluation is twofold: 1) to assess the framework's feasibility in a real-life context and 2) to preliminarily assess whether the framework provides valid results (i.e., the health indicators accurately assess the projects' health from a requirements perspective).

To achieve the first objective we applied the framework to data from three projects in the rail-automation domain (see Table 5.1). The inputs to the health assessment tool are: current requirements measures, thresholds ranges, project time ranges, and number of deadlines. Table 5.4 shows the current growth (RG), volatility (RV), design coverage (RDC), test coverage (RTC) and defect removal efficiency (DRE) measures for each project. The columns with an S prefix denote the measures for safety requirements. We were only able to acquire safety requirement measures for project 3. We note that the measures, threshold ranges, and project time ranges are also calculated by the dashboard and stored in the requirement measures database (see Figure 5.3). We used the threshold and project time ranges in Table 5.3. For simplicity, the number of project deadlines was set to one.

Table 5.4: Input measures to the health assessment tool.

Project	**RG**	**RV**	**RDC**	**RTC**	**DRE**
1	0	0	0.96	1.00	1.00
2	0	0	0.25	N/A	0.09
3	0	0.009	0.91	0.83	0.004
Project	**S-RG**	**S-RV**	**S-RDC**	**S-RTC**	**S-DRE**
1	N/A	N/A	N/A	N/A	N/A
2	N/A	N/A	N/A	N/A	N/A
3	0.53	0.57	0.95	0.55	N/A

Figure 5.6 shows a snapshot of the health assessment results for the three projects. We see that *Project 1* is in ideal shape and has a 100% green project health indicator. The health indicator for Project 2 is 40% green, 40% red, and 20% is unmeasured because, as depicted in Table 5.4, we did not have requirements-test coverage measures. The 40% red raises an alert to the internal stakeholders. Finally, we can see that *Project 3* is slightly healthier than Project 2; its health indicator is 80% green and 20% red. The safety health indicators for Project 2 raise concerns as 60% is red. The drill down below the first table in Figure 5.6 show that design coverage and defect removal efficiency for Project 2 is red. Drilling down further to the measures and project data, we'll find that design coverage is at 25% which falls into the low threshold range and given that the project is in the late stages, it is a red alert. Similarly, defect removal efficiency is at 9%, which falls into the very low threshold range with a looming deadline. Finally, Project 3 reveals an interesting observation; while growth, volatility, and test coverage health indicators are green for all project requirements, the same attributes' indicators for *safety requirements* are red. We see in Table 5.4 that growth and

volatility for safety requirements are 53% and 57%, respectively (i.e., bad threshold range). How do we explain the numbers? A closer look at the requirements documents revealed that while no new requirements were added (i.e. 0% growth) to the overall set of requirements, a large number of requirements that were initially identified as *safety irrelevant* have been, in the newest baseline, identified as *safety relevant or critical.* Additionally, because we define volatility as the sum of deletions, additions, and modifications to requirements, the number of additions was high due to this change in the safety relevance status of requirements.

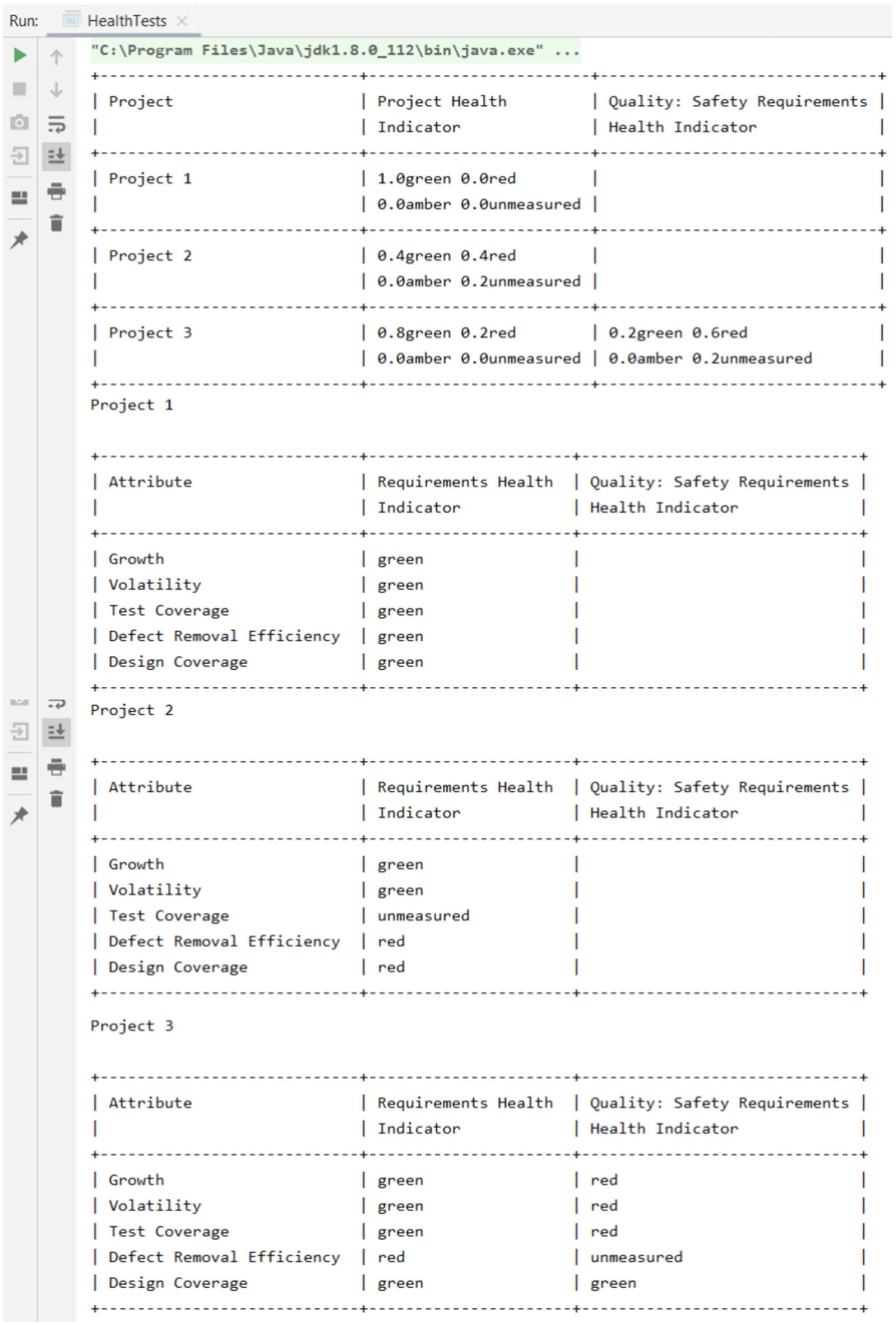
Thus, we can say that the application of the framework is feasible within the context of a systems project and that it revealed interesting requirements-related observations.

To achieve the second evaluation objective, we asked a requirements expert on Project 3 to assess the health of the five requirements attributes based on his knowledge of and experience in the project. We provided a template with a brief description of each attribute and requested that he *take into consideration other project factors that may impact their assessment such as deadlines and current project phase.* We did not provide any concrete measure for the attributes because we wanted the assessment to be based on the premise that no requirements measurement exists in the project. We then calculated the Percentage Agreement (PA) between the framework's results and the expert's assessment results.

Table 5.5 shows the assessment results of the framework and the expert. The calculated PA is 60%. We notice in Table 5.5 that the difference in assessment is for the design and test coverage requirements attributes. The expert assigned them an *amber* state because he had no concrete measures for design and test coverage, and, thus, gave it a conservative assessment to be on the safe side. However, our measurement-based framework revealed that design and test coverage are in the *high* threshold ranges and, thus, can be assigned a green health indicator.

It can be said, then, that the second evaluation objective is satisfied; the preliminary results of the health assessment framework exhibit a level of accuracy that can aid in identifying critical requirements-related problems (i.e., low requirements defect removal rates) that are detrimental to project health.

We note, however, that one must exercise caution in interpreting these results because it remains the case that the expert's interpretation of health may not fully reflect the framework's conceptualization of project health. While the framework's assessment relies strictly on quantitative measures, the expert is *gauging* project health and using his/her *gut feeling* and, thus, his/her assessment may be influenced by a plethora of non-quantitative factors.


Run: HealthTests

```
"C:\Program Files\Java\jdk1.8.0_112\bin\java.exe" ...
+---------------------------+------------------------+-----------------------------+
| Project                   | Project Health         | Quality: Safety Requirements |
|                           | Indicator              | Health Indicator            |
+---------------------------+------------------------+-----------------------------+
| Project 1                 | 1.0green 0.0red        |                             |
|                           | 0.0amber 0.0unmeasured |                             |
+---------------------------+------------------------+-----------------------------+
| Project 2                 | 0.4green 0.4red        |                             |
|                           | 0.0amber 0.2unmeasured |                             |
+---------------------------+------------------------+-----------------------------+
| Project 3                 | 0.8green 0.2red        | 0.2green 0.6red             |
|                           | 0.0amber 0.0unmeasured | 0.0amber 0.2unmeasured      |
+---------------------------+------------------------+-----------------------------+
Project 1

+---------------------------+----------------------+------------------------------+
| Attribute                 | Requirements Health  | Quality: Safety Requirements |
|                           | Indicator            | Health Indicator             |
+---------------------------+----------------------+------------------------------+
| Growth                    | green                |                              |
| Volatility                | green                |                              |
| Test Coverage             | green                |                              |
| Defect Removal Efficiency | green                |                              |
| Design Coverage           | green                |                              |
+---------------------------+----------------------+------------------------------+
Project 2

+---------------------------+----------------------+------------------------------+
| Attribute                 | Requirements Health  | Quality: Safety Requirements |
|                           | Indicator            | Health Indicator             |
+---------------------------+----------------------+------------------------------+
| Growth                    | green                |                              |
| Volatility                | green                |                              |
| Test Coverage             | unmeasured           |                              |
| Defect Removal Efficiency | red                  |                              |
| Design Coverage           | red                  |                              |
+---------------------------+----------------------+------------------------------+

Project 3

+---------------------------+----------------------+------------------------------+
| Attribute                 | Requirements Health  | Quality: Safety Requirements |
|                           | Indicator            | Health Indicator             |
+---------------------------+----------------------+------------------------------+
| Growth                    | green                | red                          |
| Volatility                | green                | red                          |
| Test Coverage             | green                | red                          |
| Defect Removal Efficiency | red                  | unmeasured                   |
| Design Coverage           | green                | green                        |
+---------------------------+----------------------+------------------------------+
```

Figure 5.6: Results of running the health assessment tool on the three projects.

Table 5.5: Framework's and expert's assessment results for project 3.

Requirements Attribute	**Framework Assessment**	**Expert Assessment**
Growth	Green	Green
Volatility	Green	Green
Design Coverage	Green	Amber
Test Coverage	Green	Amber
Defect Removal Efficiency	Red	Red

5.7 Limitations and Future Work

The first limitation of the operationalized framework is that it did not take into account a wider range of requirements attributes (e.g., requirements quality, size, and status) and project factors (e.g., team size, compliance issues, and project resources) that would give a more accurate assessment of project health. To address this shortcoming, we plan to extend the operationalized framework to take into account further requirements attributes and project factors.

The second limitation of our work is concerned with generalizability. Because the framework is heavily dependent on requirements measurement, which, in turn, depends on extensive and accurate documentation of requirements and requirements meta-data, the framework can be best utilized in contexts with an established requirements engineering process that uses extensive documentation for its requirements in databases and employs requirements metrics to manage requirements. Such a context applies to large, systems engineering projects. The framework's applicability in other contexts such as agile projects is yet unknown. Thus, we plan to evaluate the framework's usability in different contexts (e.g., agile) and application domains and to extend it according to such contexts' needs. We note, however, the research community has accepted that the lessons from one project do not necessarily fully apply to another project [Menzies and Zimmermann, 2013]. The work in this paper is but a step towards incorporating requirements into project health assessment and systematizing the process.

Finally, we are aware that the framework was evaluated in the same context that has given rise to its need. Thus, we are currently performing further validation by requesting experts from projects 1 and 2 to manually assess project health from a requirements perspective, which will strengthen our results. Moreover, we plan to further validate the framework with more experts in different organizations and application domains to assess the framework's usefulness and the accuracy of its results.

5.8 Conclusions

Practice and literature lack requirements-centric, systematized, measurement-based, and holistic health assessment methods and tools that provide internal stakeholders with high-level health indicators of a project's likelihood of success. To address this problem, we showed four contributions in this paper: 1) We presented a conceptual architecture of the health assessment framework that is built on established measurement concepts [IEEE, 2017] and which we extended to account for concepts related to project health such as critical success factors, project data, and thresholds (Section 5.4.1). This conceptual basis would allow different operationalizations of the framework in different contexts . 2) We used the conceptual architecture to operationalize the framework based on real-life systems project data from the rail-automation domain. The final output of the framework provides high-level health indicators (red-amber-green) of critical requirements attributes based on an analysis of requirements measures and project data (Section 5.4.2). The operationalized framework serves as an extendable basis for other tailored forms of the framework in other contexts. 3) We implemented the approach in a health assessment tool as part of a requirements measurement dashboard (Section 5.5). 4) We evaluated the framework's applicability in a real-life context and its ability to provide accurate health assessments that would aid manual assessments with a quantitative and systematic assessment of project health. (Section 5.6).

In summary, this work shows that there is a need for methods and tools that integrate requirements into project health assessments. Our framework demonstrates that such an endeavour is feasible and worthwhile as it supports manual and personal assessments with a quantitative basis, which, in turn, would aid in managing and monitoring large systems projects more effectively.

References

[Ahimbisibwe et al., 2015] Ahimbisibwe, A., Cavana, R. Y., and Daellenbach, U. (2015). A contingency fit model of critical success factors for software development projects: A comparison of agile and traditional plan-based methodologies. *Journal of Enterprise Information Management*, 28(1):7–33.

[Antinyan and Staron, 2017] Antinyan, V. and Staron, M. (2017). Rendex: A method for automated reviews of textual requirements. *Journal of Systems and Software*, 131:63–77.

[Bittner, 2007] Bittner, K. (2007). Measuring project health: Part one. Technical report, IBM Corporation.

[Boehm and Basili, 2001] Boehm, B. and Basili, V. R. (2001). Software defect reduction top 10 list. *Computer*, 34(1):135–137.

[Boehm, 1991] Boehm, B. W. (1991). Software risk management: principles and practices. *IEEE Software*, (January):32–40.

[Chow and Cao, 2008] Chow, T. and Cao, D. B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81(6):961–971.

[Costello and Liu, 1995] Costello, R. J. and Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems Software*, 29:39–63.

[Damian and Chisan, 2006] Damian, D. and Chisan, J. (2006). An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Transactions on Software Engineering*, 32(7):433–453.

[Fairley, 1985] Fairley, R. E. (1985). *Software engineering concepts*. McGraw Hill, New York, Montreal.

[Ferreira et al., 2011a] Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F., and Almeida, H. C. (2011a). Identifying thresholds for object-oriented software metrics. *The Journal of Systems & Software*, 85:244–257.

[Ferreira et al., 2009] Ferreira, S., Collofello, J., Shunk, D., and Mackulak, G. (2009). Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation. *Journal of Systems and Software*, 82(10):1568–1577.

[Ferreira et al., 2011b] Ferreira, S., Shunk, D., Collofello, J., Mackulak, G., and Dueck, A. (2011b). Reducing the risk of requirements volatility: findings from an empirical survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 23:375–393.

[Franch et al., 2018] Franch, X., Gómez, C., Jedlitschka, A., López, L., Martínez-Fernández, S., Oriol, M., and Partanen, J. (2018). Data-driven elicitation, assessment and documentation of quality requirements in agile software development. In Krogstie, J. and Reijers, H. A., editors, *CAiSE 2018, LNCS 10816*, Tallin, Estonia. Springer Nature.

[Génova et al., 2013] Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., and Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41.

[Hooks and Farry, 2001] Hooks, I. F. and Farry, K. A. (2001). *Customer-centered products: Creating successful products through smart requirements management*. AMACOM, New York.

[IEEE, 1992] IEEE (1992). IEEE 1061 standard for a software quality metrics methodology. Technical report, IEEE.

[IEEE, 2017] IEEE (2017). ISO/IEC/IEEE 15939:2017(E) - Systems and software engineering - Measurement Process. Technical report, ISO/IEC/IEEE.

[Jaafari, 2007] Jaafari, A. (2007). Project and program diagnostics: A systemic approach. *International Journal of Project Management*, 25(8):781–790.

[Jones, 1996a] Jones, C. (1996a). *Software systems failure and success*. International Thomson Computer Press, Boston, MA.

[Jones, 1996b] Jones, C. (1996b). Strategies for managing requirements creep. *IEEE Computer*, 29(6).

[Jones, 2000] Jones, C. (2000). *Software assessments, benchmarks, and best practices*. Addison-Wesley.

[Jones and Bonsignour, 2012] Jones, C. and Bonsignour, O. (2012). *The economics of software quality*. Addison-Wesley.

[Kerzner, 2017a] Kerzner, H. (2017a). Project health checks. Technical report, International Institute for Learning, Inc.

[Kerzner, 2017b] Kerzner, H. (2017b). *Project management metrics, KPIs, and dashboards: A guide to measuring and monitoring project performance*. John and Wiley & Sons, third edition.

[Kulk and Verhoef, 2008] Kulk, G. P. and Verhoef, C. (2008). Quantifying requirements volatility effects. *Science of Computer Programming*, 72(3):136–175.

[Lago et al., 2009] Lago, P., Muccini, H., and Van Vliet, H. (2009). A scoped approach to traceability management. *The Journal of Systems and Software*, 82:168–182.

[Malaiya and Denton, 1999] Malaiya, Y. and Denton, J. (1999). Requirements volatility and defect density. In *Proceedings 10th International Symposium on Software Reliability Engineering*, pages 285–294, Boca Raton, FL, USA. IEEE.

[Menzies and Zimmermann, 2013] Menzies, T. and Zimmermann, T. (2013). Software analytics: So what? *IEEE Software*, 30(4):31–37.

[Mohd and Shamsul, 2011] Mohd, H. N. N. and Shamsul, S. (2011). Critical success factors for software projects: A comparative study. *Scientific Research and Essays*, 6(10):2174–2186.

[Niazi et al., 2006] Niazi, M., Wilson, D., and Zowghi, D. (2006). Critical success factors for software process improvement implementation: An empirical study. *Software Process Improvement and Practice*, 11(2):193–211.

[Park et al., 2010] Park, S., Maurer, F., Eberlein, A., and Fung, T.-S. (2010). Requirements attributes to predict requirements related defects. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, pages 42–56, Toronto, ON. ACM.

[Pena and Valerdi, 2014] Pena, M. and Valerdi, R. (2014). Characterizing the impact of requirements volatility on systems engineering effort. *Systems Engineering*, 18(1):59–70.

[Philbin and Kennedy, 2014] Philbin, S. P. and Kennedy, D. A. (2014). Diagnostic framework and health check tool for engineering and technology projects. *Journal of Industrial Engineering and Management*, 7(5):1145–11166.

[Piggot and Amrit, 2013] Piggot, J. and Amrit, C. (2013). How healthy is my project? Open source project attributes as indicators of success. 48(5):30–44.

[Rosenberg et al., 1998] Rosenberg, L. H., Hammer, T. F., and Huffman, L. L. (1998). Requirements, testing, and metrics. In *15th Annual Pacific Northwest Software Quality Conference*.

[Sharma and Kaulgud, 2012] Sharma, V. S. and Kaulgud, V. (2012). PIVoT: Project insights and visualization toolkit. In Zurich, S., editor, *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM'12)*, pages 63–69. IEEE.

[Snider et al., 2018] Snider, C., Gopsill, J. A., Jones, S. L., Emanuel, L., and Hicks, B. J. (2018). Engineering project health management: A computational approach for project management support through analytics of digital engineering activity. *IEEE Transactions on Engineering Management*, 66(3):1–12.

[Staron and Meding, 2018] Staron, M. and Meding, W. (2018). *Software development measurement programs: Development, management and evolution*. Springer.

[Wagner et al., 2012] Wagner, S., Lochmann, K., Heinemann, L., Kläs, M., Trendowicz, A., Plösch, R., Seidi, A., Goeb, A., and Streit, J. (2012). The Quamoco product quality modelling and assessment approach. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, pages 1133–1142, Zurich, Switzerland. ACM.

[Winkler and von Pilgrim, 2010] Winkler, S. and von Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling*, 9:529–565.

[Zowghi and Nurmuliani, 2002] Zowghi, D. and Nurmuliani, N. (2002). A study of the impact of requirements volatility on software project performance. In *Proceedings of the 9th Asia-Pacific Software Engineering Conference (APSEC'02)*, pages 3–11, Queensland, Australia. IEEE.

Chapter 6

A Meta-Model for Requirements-Driven Information for Internal Stakeholders[1]

6.1 Introduction

Context. The requirements engineering (RE) process and resultant requirements usually inform and interact with downstream (e.g., design and testing), upstream (e.g., contract management), and side-stream (e.g., project and quality management) processes in various ways. Each of these processes involves numerous internal stakeholders (e.g., managers, developers, and architects, etc.) who, in turn, have different concerns with regard to the impact of requirements on their respective processes. In other words, the various stakeholders need different types of requirements information in order for them to manage, control, and track their respective process activities (e.g., requirements engineer: measures that track and monitor requirements growth; architect: requirement-design coverage information; systems manager: percentage of requirements dropped per release; etc.) [Buse and Zimmermann, 2012; Doerr et al., 2004; Gross and Doerr, 2012; Hess et al., 2017a]. The burden of providing this information (hereon, "requirements-driven information"), generally falls within the realm of the requirements management process [Berenbach et al., 2009; Wiegers, 2006].

Problem. To this end, we conducted an action research (AR) study in a large systems project in the rail-automation domain to derive requirements-driven information that can be used by the project's internal stakeholders (IS) (see Section 6.3.1 for AR study details). However, we found it difficult for requirements engineers to derive and provide the various internal stakeholders with the correct requirement-driven information that addresses their

[1] A version of this paper was published in [Noorwali et al., 2019].

various concerns due to a lack of understanding of: i) the type of information that can be generated from system requirements, ii) who the ISs are that would benefit from information generated from system requirements, iii) the concerns of ISs which can be addressed by providing requirement-driven information, iv) how the ISs use that information to address their various concerns, and v) the type of artifacts needed to derive the requirements-driven information. This problem is also mirrored in the scientific literature (discussed in more detail in Section 6.2).

Principle Idea. To address the problem we experienced in industry we ask the following research questions:

- **RQ1.** What are the types of entities involved in the process of providing requirements-driven information to ISs in a large systems project?
- **RQ2.** What are the relationships that exist among the entities involved in the process of providing requirements-driven information to ISs in a large systems project?

To answer the research questions, we performed a post-analysis on the data gathered from the AR study we conducted in industry (see Section 6.3.2). The result of the post-analysis is a meta-model that maps out the entities and relationships involved in providing requirements-driven information to ISs. The anticipated benefits of using the meta-model include i) control and management of processes and resources involved in providing requirement-driven information to ISs and ii) communication among ISs.

Contributions. The key contributions of this paper are:

1. Descriptions of the entities involved in providing requirements-driven information to ISs.
2. Descriptions of the relationships among the identified entities.
3. An empirically derived meta-model that combines the identified entities and relationships.
4. A discussion of the meta-model and its implications on industry and research.

Paper Structure. Section 6.2 describes related work; Section 6.3 describes the research methods; Section 6.4 presents the meta-model with a detailed description; Section 6.5 discusses the validation procedures and threats to validity; Section 6.6 discusses implications of the meta-model, and Section 6.7 concludes the paper and describes future work.

6.2 Related Work

In this section, we first discuss modelling benefits and how modelling has been used in RE. We then focus on three key issues in RE meta-models: (a) ISs and their concerns, (b) requirements-driven information, and (c) relationships among the preceding two items.

Modelling can be defined as simplification of reality [Holt et al., 2015] and has been long recognized as an important and contributory factor in the success of software projects [Berenbach et al., 2009] since it can be an effective way to understand and manage a project's complexity [Cernosek and Naiburg, 2004]. In general, models can be used to represent products [Cernosek and Naiburg, 2004], processes [Humphrey and Kellner, 1989], and activities [Humphrey and Kellner, 1989]. Modelling provides several benefits such as: (i) better understanding of an organization's or project's business needs [Humphrey and Kellner, 1989], (ii) better understanding of the application domain, (iii) improved support for system design [Cernosek and Naiburg, 2004], and (iv) aids in defining specific processes, to name a few.

In requirements engineering, modelling is used in numerous ways. One common use of modelling in RE is to model the requirements (i.e., product modelling) and associated concepts such as goals [Van Lamsweerde, 2001], stakeholders [Yu et al., 2011], and concerns [Moreira et al., 2005]. Modelling has also been used to model the requirements engineering [Berenbach et al., 2009] and associated processes such as requirements elicitation [Hickey and Davis, 2004] and requirements analysis [Jaffe et al., 1991] (i.e., process modelling). Custom requirements engineering processes have been proposed to address the demands of specific domains (e.g., safety-critical domains [Firesmith, 2004] and big data domains [Arruda and Madhavji, 2017]) and types of systems (e.g., embedded systems [Pereira et al., 2016] and systems of systems [Holt et al., 2015]). Moreover, researchers and practitioners have collaborated to propose models that aid the RE process in achieving certain goals such as legal compliance [Nekvi et al., 2011].

Similarly, in requirements management, modelling has been used to model the requirements management process [Arpinen et al., 2011]. Other models support specific requirements management tasks such as tracing [Ramesh and Jarke, 2001] and change management [Fernandes et al., 2012]. In addition, modelling RE artifacts (i.e., RE work products) has been found to facilitate interdisciplinary communication and specification work [Geisberger et al., 2006].

With respect to ISs, the literature lacks a comprehensive understanding of the types of ISs that can exist in large systems engineering projects and their concerns regarding requirements-driven information. Though the term *stakeholders* is well-known in RE, in-depth research has focused on external stakeholders (i.e., client/customer and business) and their concerns, which are usually translated into new requirements [Sarkar and Cybulski, 2002], while the concerns of ISs (e.g., project managers, testers, and architects, etc.) are rarely addressed [Hassan et al., 2013]. In the rare cases in which ISs are addressed, the problem is two-fold: 1) they focus on developer concerns only (e.g., source code defect analytics) [Hassan et al., 2013], or 2) the stakeholders are roughly divided into generic notions of "developer" and "manager" [Buse and Zimmermann, 2012].

However, our observation is that ISs and their concerns exist at a finer granularity (e.g., different types of managers, technical stakeholders, and concerns). Such a granularity has been recognized by some researchers. For example, there are studies that investigated architects' and testers' information needs in relation to requirements and requirements specifications [Gross and Doerr, 2012; Hess et al., 2017b]. Consequently, the studies propose view-based solutions that would allow testers and architects to view the requirements specification in a format that will provide them with the requirements-based information they need. We take this work further by attempting to explicate the types of stakeholders in light of their needs with regard to requirements-driven information.

When addressing requirements-driven information, we find that requirements-driven information is usually limited to requirements quality metrics (e.g. use case completeness metrics) [Costello and Liu, 1995] and basic progress metrics (e.g., number of 'complete' requirements) [Berenbach and Borotto, 2006; Wiegers, 2006] that do not specifically address the concerns of the spectrum of internal stakeholders within a project.

Finally, the relationships amongst: (i) internal stakeholders, (ii) stakeholder concerns, and (iii) requirements-driven information have not yet received significant research attention. Thus, managing these elements to derive requirements-driven information from the correct sources and providing it to the correct ISs becomes a tedious task in practice. For example, in [Doerr et al., 2004], the authors acknowledge the complexity of information in requirements engineering processes and, thus, present an information model that "captures the major information, their abstraction levels and audience in multi-project contexts. In addition, it captures responsibilities of different stakeholder roles in terms of authorship, review, approval, and change propagation" [Doerr et al., 2004]. However, the focus in this work is on *requirements documents*.

To our knowledge, a model to support the requirements management task of measuring requirements and reporting requirements relevant information to internal stakeholders does not exist. Therefore, with our meta-model we contribute to the modelling literature, particularly in requirements engineering and management.

6.3 Research Method

The meta-model presented in this paper is a result of a post analysis performed on data gathered from an AR study we conducted in industry. Figure 6.1 provides an overview of the research methods and data used in this study. The following subsections discuss the data gathering and data analysis stages in detail.

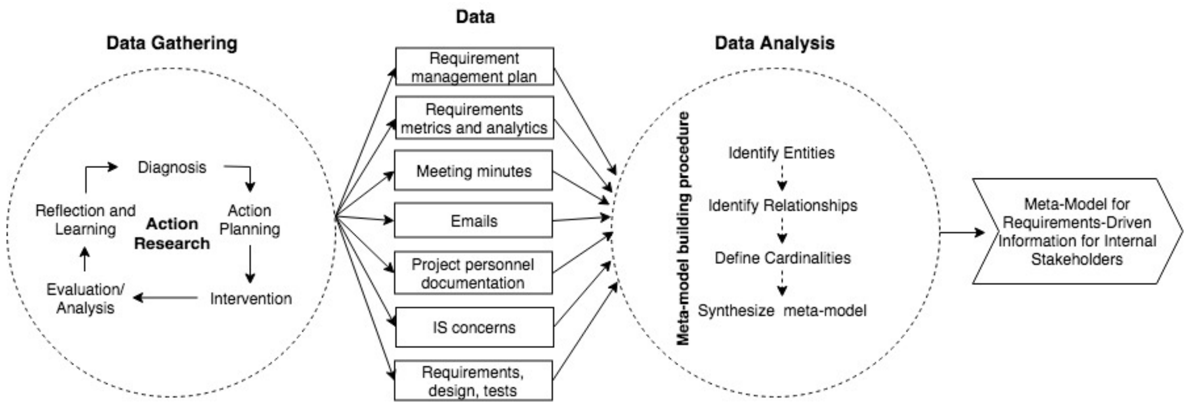


Figure 6.1: Overview of study research methods and data.

6.3.1 Data Gathering: Action Research Study

Action research (AR) is an iterative process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action planning, intervention/action taking, evaluation, and reflection/ learning [Susman and Evered, 1978], where researchers identify problems through close involvement with industrial projects, and create and evaluate solutions in an almost indivisible research activity. We note that, because the goal of the AR study was to derive requirements-driven information (not reported in this paper) to be used by the ISs, we limit our description of the AR procedure to details relevant to the meta-model and its underlying constructs.

Our AR study, which followed the described approach [Susman and Evered, 1978], was conducted in a large-scale rail automation project in a multi-national company in the United States. The overall project (i.e., program) consisted of three sub-projects, each sub-project consisted of a product that had its own set of requirements, architecture design, test cases, and engineering team. Table 6.1 shows a breakdown of the software artifacts, number of requirements, safety requirements, design objects, and test cases per project that the first author worked with. Other official project documents that were analyzed included: requirements and change management plans and project personnel documentation that describe the roles and responsibilities of the ISs involved in the projects. The project adopted a waterfall software development approach. The internal project stakeholders included: systems manager, R&D managers, test mangers, developers, architects, testers, project managers, program managers, safety managers, quality mangers, financial managers, and project operations managers.

Table 6.1: Descriptive statistics of projects.

Project	Project Duration	No. Req. Baselines	No. Reqs.	No. Safety Reqs.	No. Design Baselines	No. Design Objects	No. Test Cases
P1	73 months	54	1790	N/A	23	472	2111
P2	36 months	30	2285	N/A	4	380	N/A
P3	45 months	51	2389	923	28	827	2045

The AR study began in February 2017. The primary researcher was on-site full-time for ten months and worked with the primary industrial partner and secondary industrial participants (internal project stakeholders) in consultation with a senior researcher.

In the *diagnosis phase*, the primary researcher, primary industrial partner and senior researcher, through a series of unstructured interviews, found that a central problem in the projects' RE process is a difficulty in tracking, monitoring, and managing requirements-driven information such as requirement growth (e.g., how many requirements so far), volatility (e.g., number of changed requirements over releases), and coverage (e.g., number of requirements that have been covered by test and design) and a difficulty in accessing this information by ISs. To solve this problem, the industrial partner and researcher conducted several meetings, as part of the *action planning phase*, and decided to derive, define, and validate a set of requirements metrics and analytics that can be provided to ISs and used within the requirements management and software development processes. The requirements-driven information would include: measures on requirements size, growth, coverage, volatility, and safety requirements distribution.

In the *intervention phase*, the primary researcher, with continuous feedback from the primary industrial partner, conducted a document analysis on the requirements, design, and test documents in which the meta-data were gathered in spreadsheets and the completeness and consistency of the data were ensured. The researcher then used the gathered meta-data to define a set of metrics using GQM [Basili et al., 1994] (not reported in this paper). The measures for the different products were calculated and organized in spreadsheets and graphs. To familiarize the ISs with the derived information and to gather feedback from them, three iterations of focus groups were held. IS feedback included suggestions for new metrics and addition of descriptive information (e.g., dates) to the tables and graphs. After the three rounds of focus groups, the researcher provided the updated requirements-driven information to the ISs individually and upon request. Thus, the researcher received continuous feedback through direct engagement with the internal stakeholders and observation of the stakeholders' use of the information. Once the requirements metrics were inserted into the requirements management process, the primary researcher and industrial partner decided to add the requirements 'analytics' element by proposing a 'traffic-light' system that would provide insight into the projects' health. Such a system would utilize the derived requirements metrics in conjunction with other project artifacts such as project schedules, budget, and resources, etc. The researcher evaluated the intervention effects of the derived metrics on the requirements management and system development processes through informal discussions with the primary and secondary industrial participants and observations of the processes. Issues such as improved requirement-design coverage and improved planning of time and effort per release, etc. were noted.

As part of the *reflection and learning phase* of the AR study, the primary researcher took on the task of eliciting the challenges and lessons learned during the study, which resulted in the identification of the problems in Section 6.1 (i.e., lack of under-standing of: the types of requirements-driven information, the ISs and their concerns with regard to the requirements-driven information, IS usage of the information, and the project artifacts needed to derive the information). This, in turn, led to the research questions posed in Section 6.1. In an attempt to answer these questions, and given the availability of data from the AR study, a post-analysis was conducted to construct the meta-model, which we discuss in the following subsection.

6.3.2 Data Analysis: Meta-Model Building Procedure

To answer the research questions posed in Section 6.1, we adopted the model construction process by Berenbach et al. [Berenbach et al., 2009] as we found it to be comprehensive. Berenbach states that a holistic understanding of the domain of interest is a prerequisite before commencing a model-construction process [Berenbach et al., 2009]. Our AR study allowed us to gain first-hand and in-depth knowledge of the overall context of the RE and software development processes in the project under study. Moreover, our continuous collaboration with our industrial partner allowed for live feedback throughout the AR study and model-construction process, thus supporting incremental validation of the resultant meta-model. The key steps of the model construction process are:

Identify Entities (RQ1)

The entities were incrementally identified and added to the meta-model by analyzing the data gathered from the AR study. First, the primary researcher extracted the metrics and the IS concerns they addressed from the metric spreadsheets and GQM document that was used to define the metrics during the AR study. The ISs were identified from the project's personnel documents and from meeting minutes that were gathered from the focus groups that were conducted during the AR. The project processes were extracted from the project's requirements management and change plans. We note that, up until this point of the entity identification process, the entities were concrete project data. We then began creating abstractions of the identified entities. For example, stakeholder categories in light of their requirements-related information needs (i.e., primary technical stakeholders, regular technical stake-holders, mid-level managers, and high-level managers) were identified through analyzing the ISs' feedback and the primary researcher's correspondences with the ISs during the AR study. Specifically, the level of detail of the requirements-driven information requested by the ISs and the frequency with which they requested it were the main factors in determining these categories.

Identify Relationships among Entities (RQ2)

We identified the relationships among the entities based on organizational rules such as the relationships between software artifacts and processes and their constituents. Other relationships were identified based on the metrics derived from the AR study such as the relationship between requirements metrics and their types. Finally, some relationships were identified based on our observations of the process and interactions between various elements in the project such as the relationship between ISs and their concerns.

Synthesize the Meta-Model

The identification of the entities and their relationships occurred iteratively and in parallel. Therefore, meta-model synthesis was an ongoing process since the beginning of the meta-model building procedure. For example, when we identified three main entities at the beginning of the process (i.e., requirements metrics, ISs, and IS concerns), we added the relationships between them and further entities and relationships were iteratively added as we gained better understanding of the entities and relationships involved. Moreover, the meta-model was incrementally updated in tandem with the feedback received from the reviews by the industrial partner, senior researcher and junior researcher, which resulted in the first version of the meta-model that did not include abstraction levels. After further evaluation and feedback at a workshop session [Noorwali and Madhavji, 2018] (see Section 6.5 for validation details), the abstraction levels were added, and the entities and relationships were updated accordingly.

We adopted Berenbach's [Berenbach et al., 2009] notation, for familiarity by the sponsor's organization, to depict the meta-model elements. An entity is represented by a rectangular box with the name of the entity. A relationship is represented by a line connecting two elements with a label to indicate the type of relationship between the elements.

6.4 A Meta-Model for Requirements-Driven Information for Internal Stakeholders

The meta-model is intended to complement the organization's requirements engineering process, specifically, the requirements management process. The company's requirements engineering process consists of: requirements elicitation, analysis, validation and management. The requirements management process includes a number of activities: tracing, managing requirements workflow states, managing requirements change, deriving and reporting requirements measures and other relevant requirements-driven information; the meta-model is intended to support this latter activity.

The current version of the meta-model for requirements-driven information for ISs consists of entities and relationships organized across three abstraction levels as proposed by [Monperrus et al., 2009]. In this section we will discuss the entities, relationships among the entities, and abstraction levels. Figure 6.2 depicts the synthesized meta-model.

6.4.1 Entities

The meta-model consists of five main entities that are pertinent to the process of deriving and providing requirements-driven information to ISs:

1. **Requirements-driven information** consists of information mainly derived from requirements and requirements meta-data and that may be supported with other artifact data.
2. **Internal stakeholders** who are involved in the system development and use the requirements-driven information.
3. **Concerns** that the ISs have with regard to the requirements-driven information and that are addressed by that information.
4. **Artifacts** from which the requirements-driven information is derived.
5. **Processes** in which the ISs are involved in.

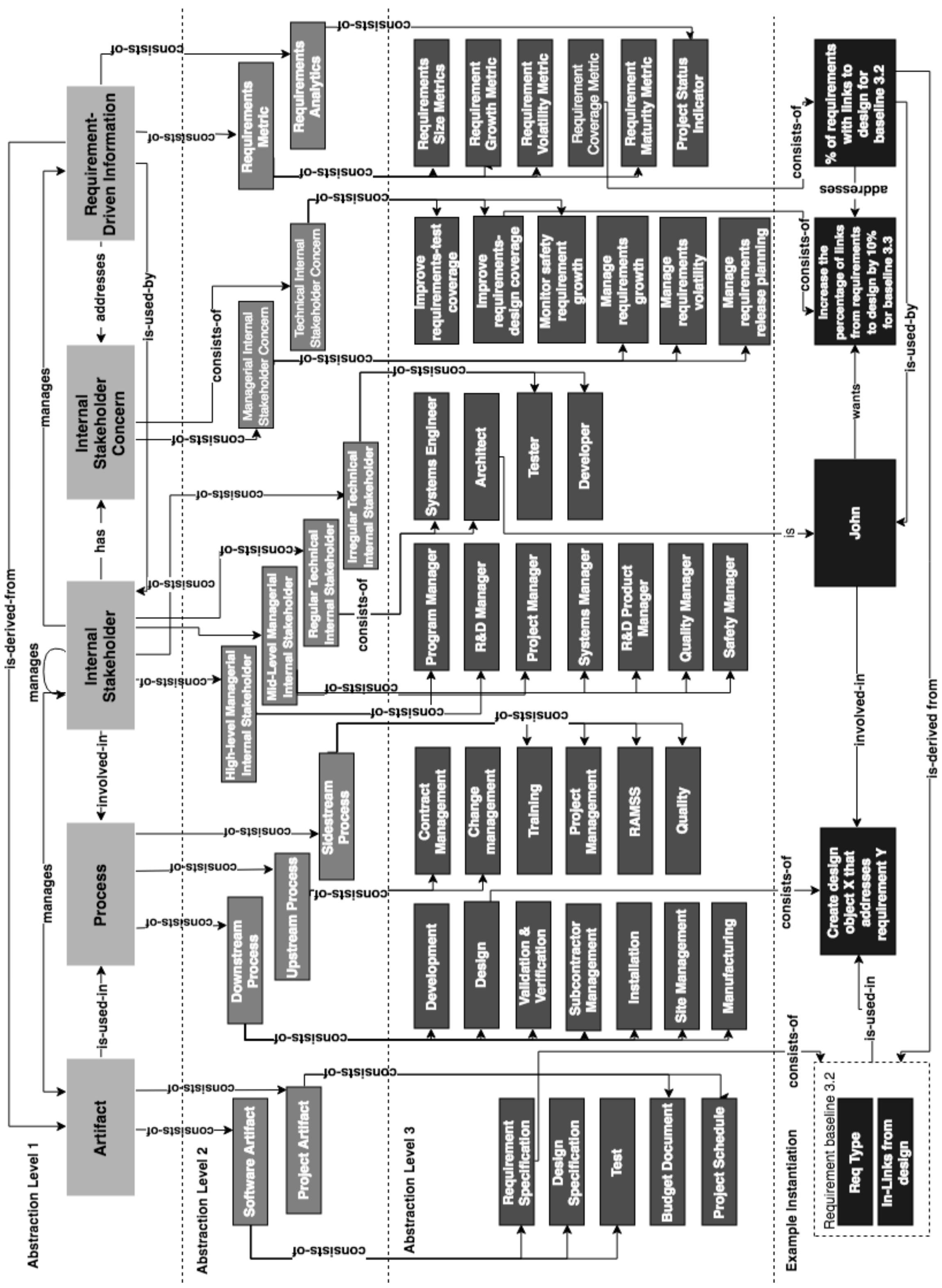

Abstraction Level 1
Abstraction Level 2
Abstraction Level 3
Example Instantiation
Requirement-Driven Information
Internal Stakeholder Concern
Internal Stakeholder
Process
Artifact
addresses
has
involved-in
is-used-in
is-used-by
manages
is-derived-from
consists-of
Requirements Metric
Requirements Analytics
Managerial Internal Stakeholder Concern
Technical Internal Stakeholder Concern
High-level Managerial Internal Stakeholder
Mid-Level Managerial Internal Stakeholder
Regular Technical Internal Stakeholder
Irregular Technical Internal Stakeholder
Sidestream Process
Upstream Process
Downstream Process
Software Artifact
Project Artifact
Requirements Size Metrics
Requirement Growth Metric
Requirement Volatility Metric
Requirement Coverage Metric
Requirement Maturity Metric
Project Status Indicator
Improve requirements-test coverage
Improve requirements-design coverage
Monitor safety requirement growth
Manage requirements growth
Manage requirements volatility
Manage requirements release planning
Systems Engineer
Architect
Tester
Developer
Program Manager
R&D Manager
Project Manager
Systems Manager
R&D Product Manager
Quality Manager
Safety Manager
Contract Management
Change management
Training
Project Management
RAMSS
Quality
Development
Design
Validation & Verification
Subcontractor Management
Installation
Site Management
Manufacturing
Requirement Specification
Design Specification
Test
Budget Document
Project Schedule
% of requirements with links to design for baseline 3.2
Increase the percentage of links from requirements to design by 10% for baseline 3.3
John
Create design object X that addresses requirement Y
Requirement baseline 3.2
Req Type
In-Links from design
is
wants
is-derived from

Figure 6.2: The meta-model for requirements-driven information for internal stakeholders.

These entities are represented at abstraction Level 1, the highest level of abstraction in the meta-model. Entities and relationships at Level 1 are abstract and generalizable enough to be applicable to any context regardless of domain, software development process, or organizational structure.

The decomposed entities constitute abstraction Level 2 of the meta-model. Entities at Level 2 are also intended to be generalizable to different contexts. However, its applicability may differ from one context to another. For example, while in a large systems project, such as ours, the distinctions between managerial and technical ISs are well defined, the differences may not be so evident in a smaller, more agile project. Thus, it is up to the project stakeholders to decide which ISs fall into which entity type. Table 6.2 consists of the entity descriptions at abstraction Level 2. Due to space limitations, we restrict our discussion to entities that are not deemed self-explanatory.

The entities at abstraction Level 2 are further decomposed into entities at abstraction Level 3. Level 3 is the project specific level in which the entities are tailored to represent the environment of a given project in a specific domain and development process. For example, requirement metrics in our study consisted of size, growth, volatility, coverage, and maturity metrics. Another project's requirements metrics may include only volatility metrics.

The same applies to other entities. Because entities at Level 3 are specific to our project, we did not include a detailed description of them. However, they can be seen in Figure 6.2 and they illustrate how the meta-model can be applied within a large systems project.

6.4.2 Relationships

The following relationships are represented in the model:

1. **is-used-by:** represents the relationship when an inanimate entity (e.g., requirements metrics) is used by an animate entity (e.g., IS) to aid in technical or managerial tasks.

2. **is-used-in:** represents the relationship between entities when an inanimate entity (e.g., artifact) is used in another inanimate entity (e.g., process) to support the definition, execution, or management of the inanimate entity it is being used in.

3. **addresses:** represents the relationship between requirements-driven information and IS concerns.

4. **consists-of:** this relationship is used when an entity (e.g., requirements-driven information) is composed of one or more of the related entities (e.g., requirements metrics and analytics).

Table 6.2: Descriptions of meta-model entities at abstraction level 2.

Entity	**Description**
Requirements Metric	A measurement derived from requirements to provide a quantitative assessment of certain requirements attributes.
Requirement Analytics	Analytics on requirements data in conjunction with other software artifacts (e.g., design, code, budget and schedule documents, etc.) that aims to gain insight about the state of the project from a requirements perspective.
High-level Managerial IS	Managerial stakeholders who manage at the project level or higher (i.e., program or regional levels) such as the program or regional R&D manager, etc.
Mid-Level Managerial IS	Managerial stakeholders who manage at the project level or lower (i.e., product level) such as test manager, product quality manager, etc.
Regular Technical IS	Technical ISs who use requirement-driven information regularly such as architects and requirements engineers.
Irregular Technical IS	Technical internal stakeholders who use requirement-driven information less frequently such as developers and testers.
Managerial IS Concern	Managerial issues that ISs care about in relation to the requirements such as estimating time and effort for a software release.
Technical IS Concern	Technical issues that ISs care about in relation to the requirements such as increasing requirement-design coverage.
Downstream Process	Activities involved in system development and initiated after the requirements engineering process such as development, design, testing, etc.
Upstream Process	Activities that are involved in system development and are initiated before the RE process such as contract/client management.
Side-stream Process	Activities involved in system development and initiated and executed alongside the RE process such as quality and project management, etc.

5. **is-derived-from:** indicates that one entity (e.g., requirements size metrics) can be defined and specified from another entity (e.g., requirements specifications).

6. **manages:** indicates that an entity (e.g., ISs) can create, add, remove, modify the related entity (e.g., software artifacts).

7. **involved-in:** indicates that an entity (e.g., IS) actively participates in the related entity (e.g., processes). The participation can be in the form of execution, management, support etc.

8. **has:** indicates that an entity (e.g., ISs) possesses one or more of the related entities (e.g., IS concerns).

The number of relationships among the entities increase as we go lower in abstraction level. This granularity provides a more detailed picture of how the decomposed entities relate to one another [Monperrus et al., 2009] in different ways. For example, at Level 1 there is one *addresses* relationship between requirements-driven information and IS concerns. The *addresses* relationships among the decomposed entities at Level 2 increase in number and are more nuanced: requirements metrics *addresses* managerial and technical IS concerns while requirement analytics *addresses* managerial IS concerns only. Figure 6.3 shows the expansion of Level 2 relationships. Similarly, the number of relationships among the decomposed entities at Level 3 increase in comparison to the relationships among the entities at Level 2. The relationships at Level 3 are project specific and thus can be tailored to project and organization rules. Due to space limitations and to preserve the readability of the model, we did not include project-specific relationships at Level 3. Finally, the relationships that cross over the abstraction boundaries are *consist-of* relationships that connect the higher-level entities with their lower-level constituents.

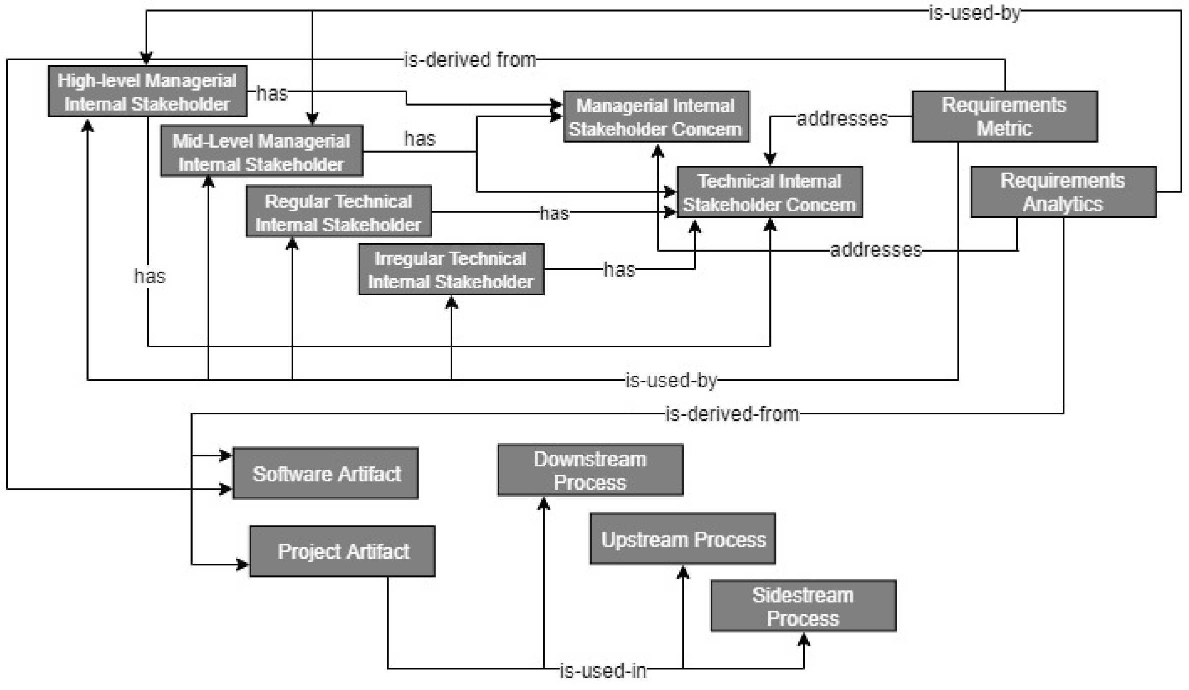

is-used-by
is-derived from
High-level Managerial Internal Stakeholder
has
Managerial Internal Stakeholder Concern
addresses
Requirements Metric
Mid-Level Managerial Internal Stakeholder
has
Technical Internal Stakeholder Concern
Requirements Analytics
Regular Technical Internal Stakeholder
has
Irregular Technical Internal Stakeholder
has
addresses
has
is-used-by
is-derived-from
Software Artifact
Downstream Process
Project Artifact
Upstream Process
Sidestream Process
is-used-in

Figure 6.3: A detailed overview of the relationships at abstraction level 2.

6.4.3 Rationale

We believe that several entity and relationship choices in the meta-model warrant a discussion of their rationale. The identification of categories of ISs and IS concerns is based on their needs regarding requirements-driven information and therefore a discussion is warranted. The meta-model separates managerial and technical internal stakeholders because they have different concerns regarding requirements measures and information, and, therefore, require different types of requirements-driven information. For example, an architect is concerned with tracking and improving requirements-architecture coverage and, thus, needs to know the number of requirements with and without links to architecture. On the other hand, a R&D product manager is concerned with estimating time and effort for a product release. Therefore, s/he needs the number of allocated requirements for a specific release. However, our experience with a large–scale systems project revealed that managerial ISs may also have technical concerns. Then, how does the separation between managerial and technical ISs affect the generated requirements measures? We observed that even in the case when a technical and managerial IS share the same technical concern, the separation between managerial and technical ISs affected the level of detail of the relevant requirements-driven information. For example, both architect and a R&D manager may want to gain insight into the state of requirements-architecture coverage. However, while the architect is interested in detailed measures (e.g., the number of requirements that do not have links to architecture per feature, per release, and per requirements baseline), the R&D manager is interested in more big-picture measures (e.g., the overall percentage of requirements that have links to architecture per requirements baseline).

As for the separation between regular and irregular technical ISs, we observed that regular technical ISs need to be frequently updated with requirements-driven information while irregular technical ISs require the relevant information less frequently. For example, the architect requires a monthly report of requirement-architecture coverage measures and in detail. On the other hand, a tester requires requirement-test cover-age measures only before a product release. Similarly, the separation between high-level and mid-level managerial ISs dictates both the frequency and level of detail of the relevant requirements information they need. These categorizations can aid the requirements engineer in knowing: what measures and information to generate and report from the requirements, to whom they should be reported, how to report it (i.e., level of detail), and when (i.e., how frequently), which, in turn, will facilitate the requirements management task of generating and reporting requirements relevant information. Finally, the rationale for separating the meta-model into abstraction levels is to facilitate the tailoring of the meta-model to different contexts, and, thus, improving its generalizability.

Example Scenario. Figure 6.2 depicts an instantiation of the model based on our project data. For example, the metric *% of requirements with links to design for requirements baseline 3.2* is derived from the project's requirements specification and uses the attributes *REQ Type* and *In-links from design* in the requirements data-base to calculate the measure. The requirements measure is used in *creating design objects that address the system requirements* that *John* (architect) is involved in and who wants to *increase requirements-design coverage by 10% for baseline 3.3.* Knowing that *John* is a regular technical IS, the measure will be reported to him in detail, which includes the percentage and absolute value of requirements-design coverage for baseline 3.2. and a list of the requirements that do not have links to design is also provided.

6.5 Meta-Model Validation

In [Shaw, 2003], Shaw states that the form of validation in software engineering must be appropriate for the type of research result. For a qualitative model, validation through evaluation demonstrates that the study results (i.e., meta-model) describes the phenomena of interest adequately [Shaw, 2003] and validation through experience shows evidence of its usefulness. Thus, the objectives of our validation are to:

1. Ensure that the meta-model adheres to the scientific principles of model building.
2. Identify missing, superfluous, and/or incorrect entities and relationships.
3. Ensure that constructs (i.e., entities and relationships) represent their correct real-world meaning.
4. Show preliminary evidence of its usefulness in practice.

To this end, the meta-model went through three phases of validation (see Table 6.3) by nine validators (see Table 6.4). The validators' areas of expertise include empirical software engineering, requirements engineering, quality and architecture, testing, software ecosystems, global and cross-organizational software development, agile methods, agent-oriented analysis, modeling, simulation, and prototyping of complex sociotechnical systems.

Table 6.3: Meta-model validation phases.

Validation Phases	**Type of Validation**	**Involved Validators**	**Method**	**Output**
Phase 1	Evaluation	V1, V2, V3	Expert opinion	Version 1 of the meta-model (not included in paper)
Phase 2	Evaluation	V1, V4, V5, V6	Live study at workshop	Version 2 of the meta-model (included in paper)
Phase 3	Evaluation, experience	V7, V8, V9	Expert opinion	Evidence of meta-model usefulness

Table 6.4: Profiles of meta-model validators.

Validator	**Research Experience**	**Industry Experience**	**Involved in Studied Project?**
V1 Researcher	40 years	33 years of industry collaboration	No
V2 Practitioner	6 years	8 years	Yes
V3 Researcher	5 years	4 years	No
V4 Researcher	16 years	10 years of industry collaboration	No
V5 Researcher	44 years	30 years of industry collaboration	No
V6 Researcher	25 years	11 years	No
V7 Practitioner	2 years	17 years	Yes
V8 Practitioner	9 years	8 years	Yes
V9 Practitioner	6 years	5 years	No

6.5.1 Phase 1

V1 reviewed the meta-model for the soundness of its entities and relationships. He also brought to our attention the notion of *change* in the meta-model. That is, who makes the changes to requirements metrics, software artifacts and stakeholders? This is in line with Berenbach's approach in which he states that the following questions must be asked when building a meta-model [Berenbach et al., 2009]: Who creates the entities? Who modifies them? How do they become obsolete? This feedback from **V1** resulted in the addition of the *manages* relationship between: ISs and metrics, ISs and software artifacts, ISs and ISs (see Section 6.4 and Figure 6.2).

V2 is the main requirements management figure in the project that we conducted our AR study. He manages the RE processes for all the products in the rail-automation project. He, therefore, is the most knowledgeable internal stakeholder on the RE processes. His validation consisted of feedback on the soundness of the meta-model constructs (i.e., entities and relationships) and ensured that the entities and relationships represented the project accurately. **V3** also reviewed the technical aspects of the meta-model to ensure the correctness of the meta-model. He also aided the first author in identifying proper relationship labels and reviewing the semantics of the meta-model.

6.5.2 Phase 2

Phase 2 consisted of an exploratory, interactive study at EmpiRE'18 [Noorwali and Madhavji, 2018] in which the meta-model from Phase 1 was presented and explained to the audience. The participants were given questions to validate the meta-model, and then asked to write their answers on post-it notes that were pinned to their designated areas on the wall (see Appendix A for study details). Twenty-seven answers were provided in total and were used to enhance the meta-model. The main piece of feedback from the live study was the suggestion to divide the meta-model into abstraction levels.

6.5.3 Phase 3

Phase 3 is ongoing and consists of validating the meta-model for its usefulness in practice. To this end, we have sent out the meta-model to practitioners to gather their feedback on its usefulness (see Appendix B for validation form). We have received feedback from three practitioners (V7, V8, V9), who asserted that the meta-model would be useful in practice with some modifications.

V7, who has managed the project's quality management processes and is involved in the system architecture, says the meta-model would be very useful in managing the requirement-driven information that can be generated and disseminated among ISs. However, he suggests:

> *"This information get captured in modeling tools and thus tied to the system structure as opposed to chapters in a document for increased usability."*

V8 is from an external organization and states:

> *"I think the key are stakeholders. So taking the perspective of "WHO does/needs/provides WHAT?", this model would be a great way to elaborate what the stakeholder descriptions/roles are (for the internal stakeholders, and secondarily for the customer / upper management). In that respect, this model is a mental model that is used after having done stakeholder discovery (e.g., with the onion model) and gives some tools while documenting the stakeholder roles (e.g., when determining the importance influence)."*

V9 is also an external validator and states:

> *"I think the meta-model is useful for practice overall, as it helps taking the different types of stakeholders into account, not omitting important ones, it helps thinking of the different processes impacted and various types of concerns and artifacts, thus not missing out important things. However, I did not vote for "Strongly agree", as I think its success in practice strongly depends on its acceptance by all crucial stakeholders involved, which can be highly challenging in organizations working under time pressure, very profit-driven, generally ad-hoc and without too much focus on following given models, even if they believe they could be useful."*

> *"As mentioned above, I think the model can help in covering all important aspects and not missing what matters. It helps structuring reasoning, as well as providing a clear, structured way for presenting input to all involved stakeholders. Although rather project-specific, Abstraction Level 3 is also useful to give the user an idea of what could fit on this level (e.g., different roles) and one can use this as a starting point or inspiration when adapting the model to his/her specific context. I would use the model to show the different inter-relations to my internal stakeholders, but I would probably adapt what I show depending on my target audience: the high management (CxO) will not have the patience to look into the details of a complex representation, but my RD teams would be happy to dig deeper. So I'd probably build one thorough model for myself and then adapt it to fit my meetings."*

Thus, phase three provides preliminary evidence for the anticipated practical benefits discussed in Section 6.1. V8 also suggested the replacement of the monochrome color scheme with different colors to facilitate reading and comprehension of the meta-model.

6.5.4 Threats to Validity

We discuss the study validity threats and how we mitigated them according to Runeson and Host's guidelines [Runeson and Höst, 2009].

Internal Validity

Internal validity is concerned with the validity of causal relationships, typically in scientific experiments. Given that our study objective does not include investigation of causal relationships, this threat is not relevant to our study.

External Validity

External validity is concerned with the generalizability of the results to other contexts. The meta-model is based on the AR study conducted within the safety-critical, transportation domain, which may limit the meta-model's generalizability. Thus, readers must interpret and reuse the results in other contexts with caution. Despite this limitation, the results constitute an important data-point for making scientific progress. Further validation of the meta-model in different domains and project sizes is encouraged in order to improve its generalizability.

Construct Validity

Construct Validity concerns the operationalized constructs of the study in that whether or not they accurately represent the real-world phenomena. It is possible that some meta-model entities (e.g., stakeholder concerns and metrics, etc.) might not have been captured accurately by the researcher. In order to minimize this threat, we validated the model constructs with our industrial partner and analyzed them against official project documentation to ensure that the constructs accurately reflect their real-world counterparts. In addition, given that the meta-model was not the main goal of the AR study, there is a risk that important data is missing from the meta-model. This risk was mitigated by obtaining feedback from a variety of sources on the meta-model entities and relationships during the workshop (see Appendix A for workshop details).

Reliability

Reliability is concerned with the degree of repeatability of the study. The AR study followed AR principles for software engineering [Santos and Travassos, 2011] to ensure rigor during the study. In addition, the AR and meta-model creation processes were documented to ensure traceability and analysis. Although a level of subjectivity is inevitable during the meta-model development process, our continuous involvement with our industrial partners and researchers inside and outside of the study helps to mitigate this threat.

6.6 Implications

6.6.1 Implications for Practice

The meta-model can aid in aligning internal stakeholder concerns with requirements-driven information that can be generated within the project [Humphrey and Kellner, 1989]. It can also be an effective tool for enabling effective communication as well as controlling project complexity [Humphrey and Kellner, 1989]. In our case, the complexity is the network of numerous internal stakeholders, stakeholder concerns, requirement metrics and analytics, downstream, upstream and side-stream processes, and a web of interactions amongst them. Therefore, mapping out the numerous elements and the relationships amongst them will equip requirement engineers with the understanding needed to effectively control and manage the requirements-driven information they are required to provide [Humphrey and Kellner, 1989] and communicate to the right people (see Phase 3 of validation in Section 6.5). The meta-model could also aid incoming personnel (e.g., new requirements engineers) in understanding this complex web of interactions, which, in turn, will help them in their requirements management tasks.

The meta-model can also serve as a stepping-stone toward operationalizing the entities and relationships in the meta-model in the form of a tool (e.g., dashboard) that could aid practitioners in the requirements management process by implementing features inspired by the meta-model (see Phase 3 of validation in Section 6.5).

6.6.2 Implications for Research

The importance of requirements-driven information for internal stakeholders has been recognized by researchers [Gross and Doerr, 2012; Hassan et al., 2013]. As discussed earlier, some research efforts have targeted architects' and testers' information needs in relation to requirements and requirements specifications [Gross and Doerr, 2012; Hess et al., 2017b]. Our attempt to explicate the types of stakeholders in light of their needs with regard to requirements-driven information can open further avenues for research. Specifically, further research can be conducted to explore questions addressing IS information needs with regard to requirements. Such questions could include: what are the types of ISs in an agile environment? What are their information needs with regard to requirements in an agile environment? In addition to requirements metrics and analytics, what other types of information can be generated from requirements and that can benefit internal stakeholders in their processes?

6.7 Conclusions and Future Work

Requirements are an information-rich software artifact that has the potential to provide ISs with information that can guide their respective processes. However, little is known about the types of ISs in light of their requirements-information needs, the information that can be generated from requirements, and how this information is used by ISs, all of which complicates the requirements management process. Based on empirical data that we gathered and analyzed from an AR study conducted in a large-scale rail automation project (Section 6.3.1), we identified the main entities and relationships involved in providing requirement-driven information, which we assembled into a meta-model (Section 6.3.2). The empirically derived meta-model depicts the internal stakeholders, internal stakeholder concerns, requirements-driven information, artifacts, processes, and relationships among them at three abstraction levels (Section 6.4).

Our preliminary validation shows that the meta-model aids in understanding the complex network of entities and relationships involved in providing requirements-driven information to internal stakeholders (Section 6.5). More specifically, the explicit identification of the types of internal stakeholders and their needs in relation to requirement-driven information could facilitate: (i) communication among internal stakeholders and (ii) proper identification and presentation of requirement-driven information for the correct internal stakeholders.

For future work, we intend to extend the meta-model to include cardinalities, which will provide a more accurate representation of a project's rules and policies. For example, only one IS (i.e., requirements engineer) manages the requirements-driven information. This cardinality is a representation of the current project practices. Therefore, upon reading the meta-model, one would know that one person is in charge of managing the various requirements-driven information and so appropriate interpretation is facilitated. We also plan to incorporate the meta-model into the organization's requirements management plan to validate it empirically for its practicality, usefulness, and benefits within the project.

References

[Arpinen et al., 2011] Arpinen, T., Hamalainen, T. D., and Hannikainen, M. (2011). Meta-model and UML profile for requirements management of software and embedded systems. *EURASIP Journal on Embedded Systems*, 2011.

[Arruda and Madhavji, 2017] Arruda, D. and Madhavji, N. H. (2017). Towards a requirements engineering artefact model in the context of big data software development projects: Research in progress. In *IEEE International Conference on Big Data*, pages 2314–2319. IEEE.

[Basili et al., 1994] Basili, V., Caldiera, G., and Rombach, H. (1994). Goal question metric approach. *Encyclopedia of Software Engineering*, 1:98–102.

[Berenbach and Borotto, 2006] Berenbach, B. and Borotto, G. (2006). Metrics for model driven requirements development. In *28th International Conference on Software Engineering (ICSE 2006)*, pages 445–451, Shanghai, China. ACM.

[Berenbach et al., 2009] Berenbach, B., Paulish, D. J., Kazmeier, J., and Rudorfer, A. (2009). *Software and systems requirements engineering in practice*. McGraw Hill.

[Buse and Zimmermann, 2012] Buse, R. P. L. and Zimmermann, T. (2012). Information needs for software development analytics. In *34th International Conference on Software Engineering (ICSE 2012)*, pages 987–996, Zurich, Switzerland. IEEE.

[Cernosek and Naiburg, 2004] Cernosek, G. and Naiburg, E. (2004). The value of modeling. Technical report, IBM.

[Costello and Liu, 1995] Costello, R. J. and Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems Software*, 29:39–63.

[Doerr et al., 2004] Doerr, J., Paech, B., and Koehler, M. (2004). Requirements engineering process improvement based on an information model. In *Proceedings of the 12th IEEE International Conference on Requirements Engineering (RE'04)*, pages 70–79, Vancouver, Canada. IEEE.

[Fernandes et al., 2012] Fernandes, J., Silva, A., and Henriques, E. (2012). Modeling the impact of requirements change in the design of complex systems. In Aiguier, M., Caseau, Y., Krob, D., and Rauzy, A., editors, *Proceedings of the 3rd International Conference on Complex Systems Design & Management (CSD&M 2012)*, pages 151–164, Paris, France. Springer.

[Firesmith, 2004] Firesmith, D. (2004). Engineering safety requirements. *The Journal of Object Technology*, 3(3):27–42.

[Geisberger et al., 2006] Geisberger, E., Broy, M., Berenbach, B., Kazmeier, J., Paulish, D., and Rudorfer, A. (2006). Requirements engineering reference model. Technical report, Institut fur Informatik der Technischen Universitat Munchen.

[Gross and Doerr, 2012] Gross, A. and Doerr, J. (2012). What do software architects expect from requirements specifications? Results of initial explorative studies. In *1st IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks 2012)*, pages 41–45, Chicago, Illinois. IEEE.

[Hassan et al., 2013] Hassan, A. E., Hindle, A., Runeson, P., Shepperd, M., Devanbu, P., and Kim, S. (2013). Roundtable: What's next in software analytics. *IEEE Software*, 30(4):53–56.

[Hess et al., 2017a] Hess, A., Diebold, P., and Seyff, N. (2017a). Towards requirements communication and documentation guidelines for agile teams. In *25th IEEE International Requirements Engineering Conference Workshops, (REW 2017)*, pages 415–418. IEEE.

[Hess et al., 2017b] Hess, A., Doerr, J., and Seyff, N. (2017b). How to make use of empirical knowledge about testers' information needs. In *25th IEEE International Requirements Engineering Conference Workshops (REW 2017)*, pages 327–330. IEEE.

[Hickey and Davis, 2004] Hickey, A. M. and Davis, A. M. (2004). A unified model of requirements elicitation. *Journal of Management Information Systems*, 20(4):65–84.

[Holt et al., 2015] Holt, J., Perry, S., Payne, R., Bryans, J., Hallerstede, S., and Hansen, F. O. (2015). A Model-Based Approach for Requirements Engineering for Systems of Systems. *Systems Journal, IEEE*, 9(1):252–262.

[Humphrey and Kellner, 1989] Humphrey, W. S. and Kellner, M. I. (1989). Software process modeling: Principles of entity process models. In *11th International Conference on Software Engineering (ICSE 1989)*, pages 331–342, Pittsburgh, PA. ACM.

[Jaffe et al., 1991] Jaffe, M., Leveson, N., Heimdahl, M., and Melhart, B. (1991). Software requirements analysis for real-time process-control systems. *IEEE Transactions on Software Engineering*, 17(3):241–258.

[Monperrus et al., 2009] Monperrus, M., Beugnard, A., and Champeau, J. (2009). A definition of "abstraction level" for metamodels. In *16th Annual International IEEE Conference and Workshop on Engineering of Computer Based Systems (ECBS 2009)*, pages 315–320. IEEE.

[Moreira et al., 2005] Moreira, A., Araújo, J., and Rashid, A. (2005). A concern-oriented requirements engineering model. *Advanced Information Systems Engineering*, 3520:293–308.

[Nekvi et al., 2011] Nekvi, R., Ferrari, R., Berenbach, B., and Madhavji, N. H. (2011). Towards a compliance meta-model for system requirements in contractual projects. In *4th International Workshop on Requirements Engineering and Law, RELAW 2011*, pages 74–77, Trento, Italy. IEEE.

[Noorwali and Madhavji, 2018] Noorwali, I. and Madhavji, N. H. (2018). A domain model for requirements-driven insight for internal stakeholders: A proposal for an exploratory interactive study. In *7th IEEE International Workshop on Empirical Requirements Engineering (EmpiRE 2018)*, pages 32–36. IEEE.

[Noorwali et al., 2019] Noorwali, I., Madhavji, N. H., Arruda, D., and Ferrari, R. (2019). Towards a meta-model for requirements-driven information for internal stakeholders. In Knauss, E. and Goedicke, M., editors, *25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2019)*, volume 1, pages 262–278, Essen, Germany. Springer Nature.

[Pereira et al., 2016] Pereira, T., Albuquerque, D., Sousa, A., Alencar, F., and Castro, J. (2016). Towards a metamodel for a requirements engineering process of embedded systems. In *VI Brazilian Symposium on Computing Systems Engineering*, pages 93–100, Joao Pessoa, Brazil. IEEE.

[Ramesh and Jarke, 2001] Ramesh, B. and Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93.

[Runeson and Höst, 2009] Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164.

[Santos and Travassos, 2011] Santos, P. S. M. d. and Travassos, G. H. (2011). Action research can swing the balance in experimental software engineering. *Advances in Computers*, 83:205–276.

[Sarkar and Cybulski, 2002] Sarkar, P. K. and Cybulski, J. L. (2002). Aligning system requirements with stakeholder concerns: Use of case studies and patterns to capture domain expertise. In *7th Australian Workshop on Requirements Engineering (AWRE 2002)*, pages 67–82.

[Shaw, 2003] Shaw, M. (2003). Writing good software engineering research papers. In *25th International Conference on Software Engineering (ICSE 2003)*, pages 726–736, Portland, Oregon. IEEE.

[Susman and Evered, 1978] Susman, G. and Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative Science Quarterly*, 23(4):582–603.

[Van Lamsweerde, 2001] Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: a guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 249–262, Toronto, Canada. IEEE.

[Wiegers, 2006] Wiegers, K. E. (2006). *More about software requirements: Thorny issues and practical advice*. Microsoft Press, Washington.

[Yu et al., 2011] Yu, E., Giorgine, P., Maiden, N., and Mylopoulos, J. (2011). *Social modeling for requirements engineering*. MIT Press.

Chapter 7

R-Pulse: A Requirements Dashboard Prototype

7.1 Introduction

Automated In-Process Software Engineering Measurement and Analysis (AISEMA) is the non-invasive data collection and analysis of data in software engineering processes for the purpose of measurement [Johnson, 2007; Coman et al., 2009]. Automating the measurement process has been found to be a key requirement for the success of software programs [Daskalantonakis, 1992; Pfleeger, 1993; Hall and Fenton, 1997; Iversen and Mathiassen, 2000; Ebert and Dumke, 2007; Staron and Meding, 2018]. Automating data collection, analysis, visualization, and reporting mechanisms of measurement reduces the overhead associated with data collection, makes analysis and dissemination of feedback easier and more accurate, and reduces human error in data collection [Gopal et al., 2002]. In particular, measurement-driven dashboards provides the foundation for effective and efficient management of organizations and projects that develop large-scale systems [Selby, 2009].

While many academic and commercial measurement tools and dashboards have been proposed to support software processes [Sillitti et al., 2003; Johnson et al., 2003; Sharma and Kaulgud, 2012; López et al., 2018], most focus on the development phases and are based on code-centric metrics with little to no support for requirements and requirement measurement. Moreover, current requirements management tools do not provide advanced measurement functionalities and are limited to primitive counts of requirements and visualizations.

Thus, in an attempt to fill this gap and facilitate the use of the measurement concepts presented in the previous chapters, we present *R-Pulse*, a requirements-focused dashboard that gathers, processes, analyzes, and visualizes requirements measures and indicators. Particularly, R-Pulse has a three-tier architecture and provides the following features: data gathering and management, data analysis, and data visualization and navigation. It is intended to be used by internal stakeholders (e.g., requirements engineers, architects, and product managers) of a systems project to aid their process-related tasks. The internal stakeholders can choose from a selection of requirements metrics, requirements health indicators and visualization techniques that best address their respective concerns.

In this paper, we first discuss the requirements measurement challenges faced in the collaborating organization in the absence of tool support in Section 7.2. We then present R-Pulse and discuss its main features and architecture in Sections 7.3.1 and 7.3.2, respectively. Section 7.4 describes the dashboard development process followed by a survey and comparison of related work in Section 7.5. We then discuss the dashboard's limitations in Section 7.6 and finally Section 7.7 concludes the paper and discusses future work.

7.2 Challenges to Requirements Measurement

While the concepts in Chapters 3, 5, and 6 lay the theoretical foundations for the requirements measurement process, most requirements management tools (e.g., DOORS) do not include advanced requirements measurement features, which leads to a manual or semi-manual measurement process. Thus, the measurement process becomes even more cumbersome in the context of large system projects that has large numbers of requirements, many internal stakeholders, and large numbers of metrics to be implemented for projects, to name a few. This added effort is a main barrier to the adoption of metrics in software projects in general [Kerzner, 2017].

In this subsection, we describe the requirements measurement challenges that emerged within the context of the collaborating company (see Chapter 2 for company details) upon attempting to implement a requirements measurement program. Table 7.1 lists the main requirements challenges we faced, the way they are dealt with in the absence of a dashboard, and how R-Pulse addresses each challenge.

Table 7.1: Requirements measurement challenges in the absence of tool support.

	Challenge	Status without R-Pulse	Solution in R-Pulse
C1	**Multiple projects, numerous requirements baselines, and metrics.** Numerous requirements baseline were generated regularly for each project. In turn, each requirements baseline entailed updating the requirements measures to reflect the newly generated baselines.	An individual is designated for exporting the new requirements baselines for multiple projects and integrating them with the metrics' spreadsheets to update the measures, which became a tedious and time-consuming task. Especially with a growing number of metrics and projects being added.	The dashboard reduces the time and effort needed to update the requirements measures by automating the computation process.
C2	**Different requirements-driven information for different internal stakeholders.** Each project had a large number of internal stakeholders and each required different requirements-driven information to be provided to him/her.	When the metrics' spreadsheets are updated, disseminating the measures of interest to the correct internal stakeholders led to many mutations of the measures' report that was tailored to each stakeholder's needs. This process required a significant amount of added time and effort.	The dashboard allows each internal stakeholder to see the measures and indicators they are interested in through various filtering options (e.g., requirements attributes, dates, baseline numbers)
C3	**Different visualizations for different internal stakeholders.** While some internal stakeholders were only interested in high-level visualizations of the requirements measures (e.g., program manager), others were interested in the raw numbers.	The measurement reports were tailored to meet the various demands of the internal stakeholders across the different projects. For example, bar and pie charts were created for the managers, while detailed measurement reports were sent to the developers and architects.	Using the concepts in Chapter 6, we developed the dashboard to address the different internal stakeholders' visualization needs. Thus, internal stakeholders could select among the different visualization options that address their specific needs.
C4	**Different reporting frequency.** Some internal stakeholders (e.g., architect) required weekly reports of the requirements measures while monthly reports sufficed for others (e.g., R&D manager)	Due to the tedious process of updating the metrics, updating the spreadsheets was limited to once a month. In the event an internal stakeholder had an urgent need for an updated report (i.e., before a product release) before the monthly deadline, a report was generated for him/her.	R-Pulse addresses this problem by providing constant, individual access to the updated requirements-driven information.
C5	**Advanced analysis of measures and project data.** There was a need for advanced analysis of the requirements measures along with project data (see Chapter 5).	Such complex analyses were impossible without some form of automation.	R-Pulse automates the analysis of the requirements measures in conjunction with project data and visualizes the resulting indicators for the internal stakeholder.

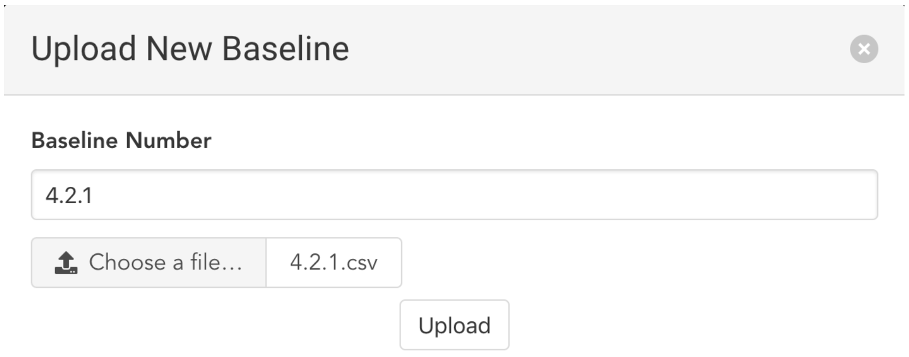
7.3 The Requirements Dashboard Prototype: R-Pulse

R-Pulse builds upon the theoretical foundations in Chapters 3, 5, and 6. The purpose of the dashboard is to automate the process of providing requirements-driven information to the internal stakeholders. It particularly addresses the measurement challenges discussed in Table 7.1. This automation, in turn, would reduce the time and effort needed for the measurement process. In addition, the dashboard would provide easy and continuous access to requirements-driven information, which, in turn, would aid internal stakeholders in making requirements-related decisions in their respective tasks (e.g., architecting, development, and release planning). Below we discuss the dashboard's main features and architecture.

7.3.1 Main Features

The dashboard's main features are:

Data Gathering and Management The main data input to the current version of the dashboard are spreadsheet exports of requirements-design, test, and defect baselines. The exports are uploaded through the dashboard's graphical user interface (GUI) and stored in the database. Figure 7.1 shows a screenshot of the *upload spreadsheet* page, which is then saved into a database.

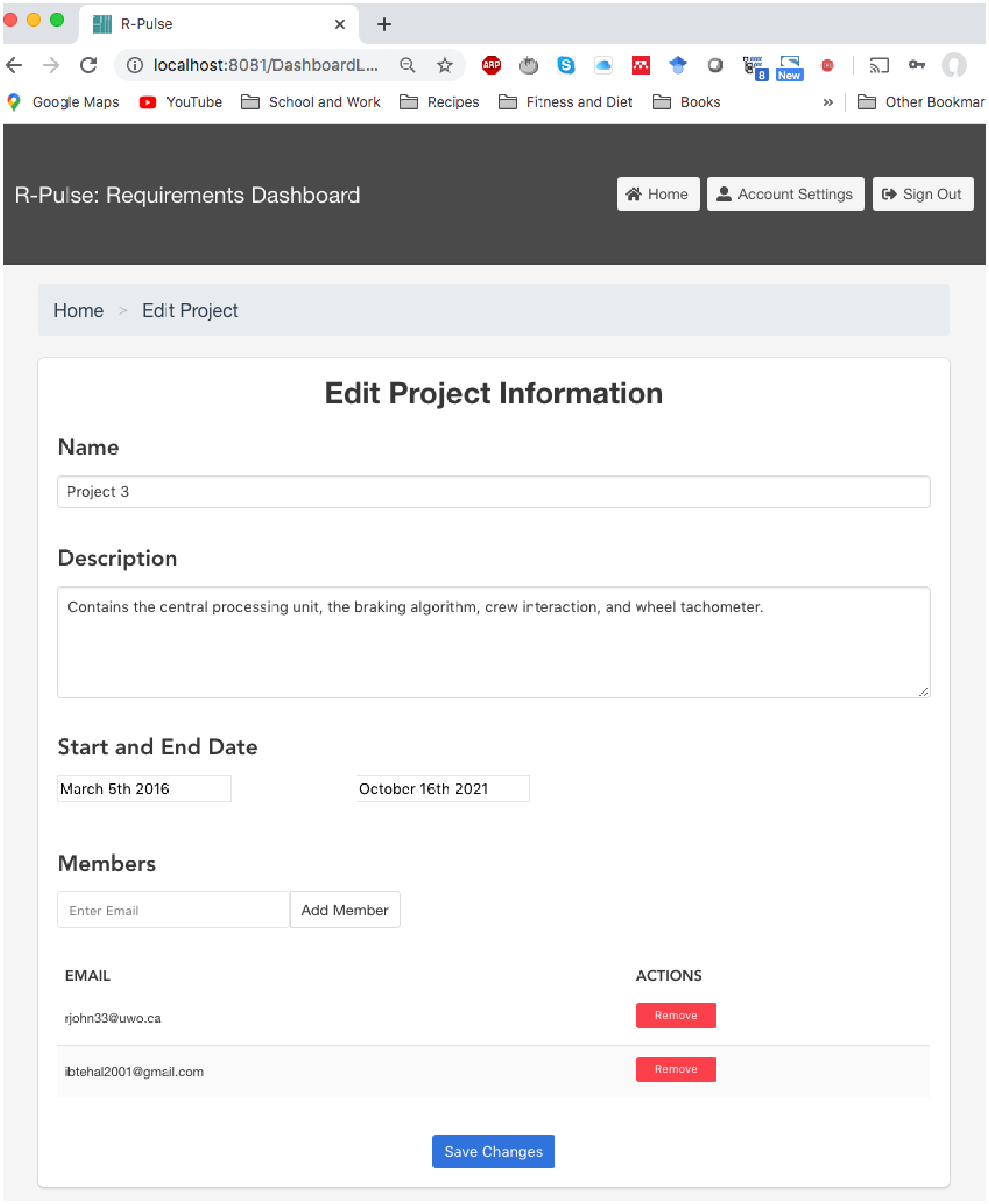

Upload New Baseline

Baseline Number

4.2.1

Figure 7.1: Upload spreadsheet page in R-Pulse.

In addition, the dashboard includes functionality to enter data for projects, artifacts (i.e., requirements, design, test, and defects), attributes, metrics, levels, and users. The manually input data and the data in the spreadsheets are then stored in database. Figures 7.2 and 7.3 show screenshots of the pages for entering project and artifact data. A project consists of artifacts (e.g., requirements documents, requirements defect documents, and design documents, etc.), team members, and start and end dates. An artifact (e.g., requirements specification document—SRS) consists of uploaded baseline, measurable attributes (e.g., size, growth, and volatility), metric levels (e.g., feature, release, and safety).



R-Pulse: Requirements Dashboard

Edit Project Information

Name

Project 3

Description

Contains the central processing unit, the braking algorithm, crew interaction, and wheel tachometer.

Start and End Date

March 5th 2016

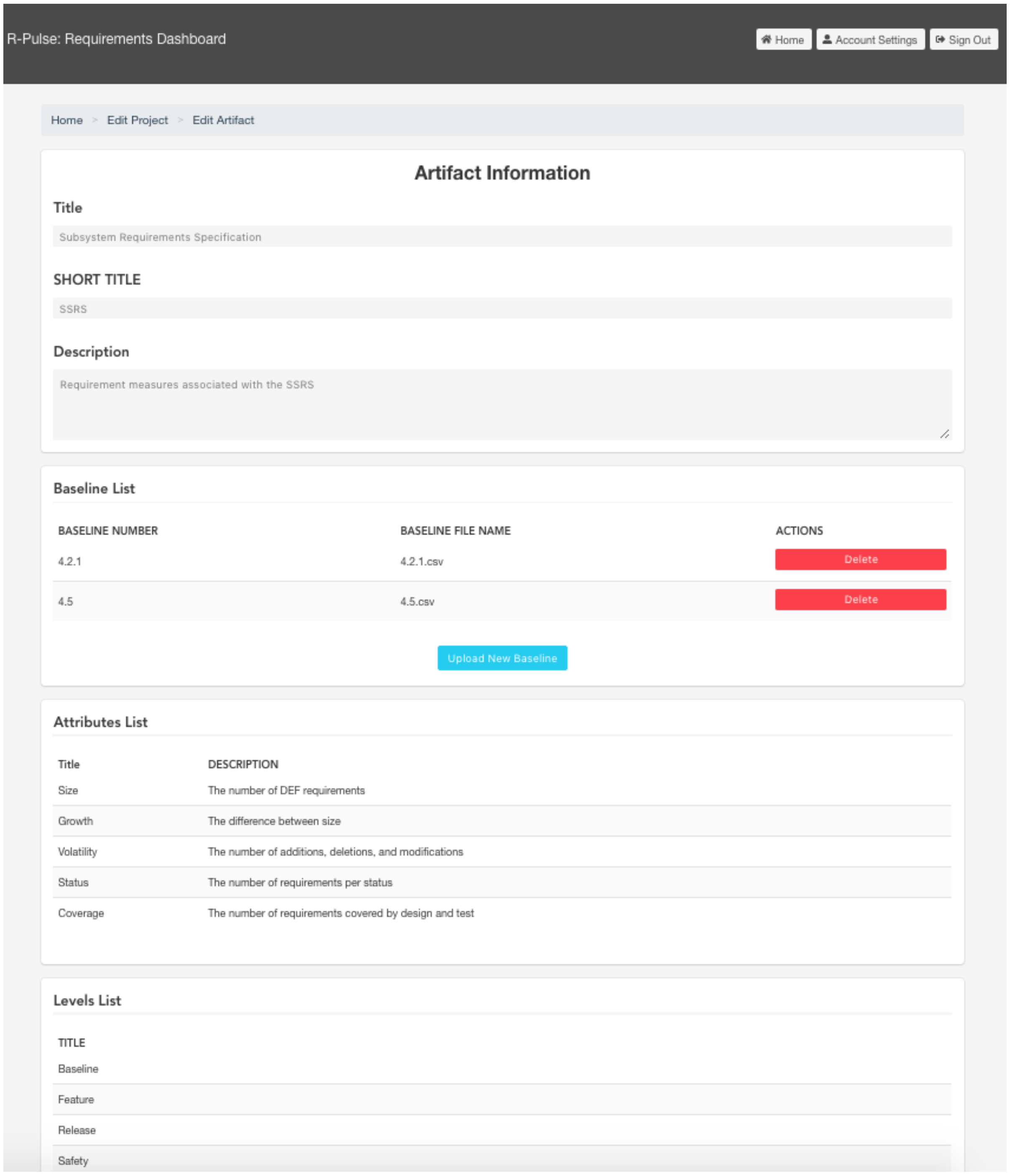
October 16th 2021

Members

Enter Email

EMAIL	ACTIONS
rjohn33@uwo.ca	Remove
ibtehal2001@gmail.com	Remove

Figure 7.2: Edit project page in R-Pulse.


R-Pulse: Requirements Dashboard
Home
Account Settings
Sign Out
Home > Edit Project > Edit Artifact
Delete
Delete
Upload New Baseline

Artifact Information

Title

Subsystem Requirements Specification

SHORT TITLE

SSRS

Description

Requirement measures associated with the SSRS

Baseline List

BASELINE NUMBER	BASELINE FILE NAME	ACTIONS
4.2.1	4.2.1.csv	Delete
4.5	4.5.csv	Delete

Attributes List

Title	DESCRIPTION
Size	The number of DEF requirements
Growth	The difference between size
Volatility	The number of additions, deletions, and modifications
Status	The number of requirements per status
Coverage	The number of requirements covered by design and test

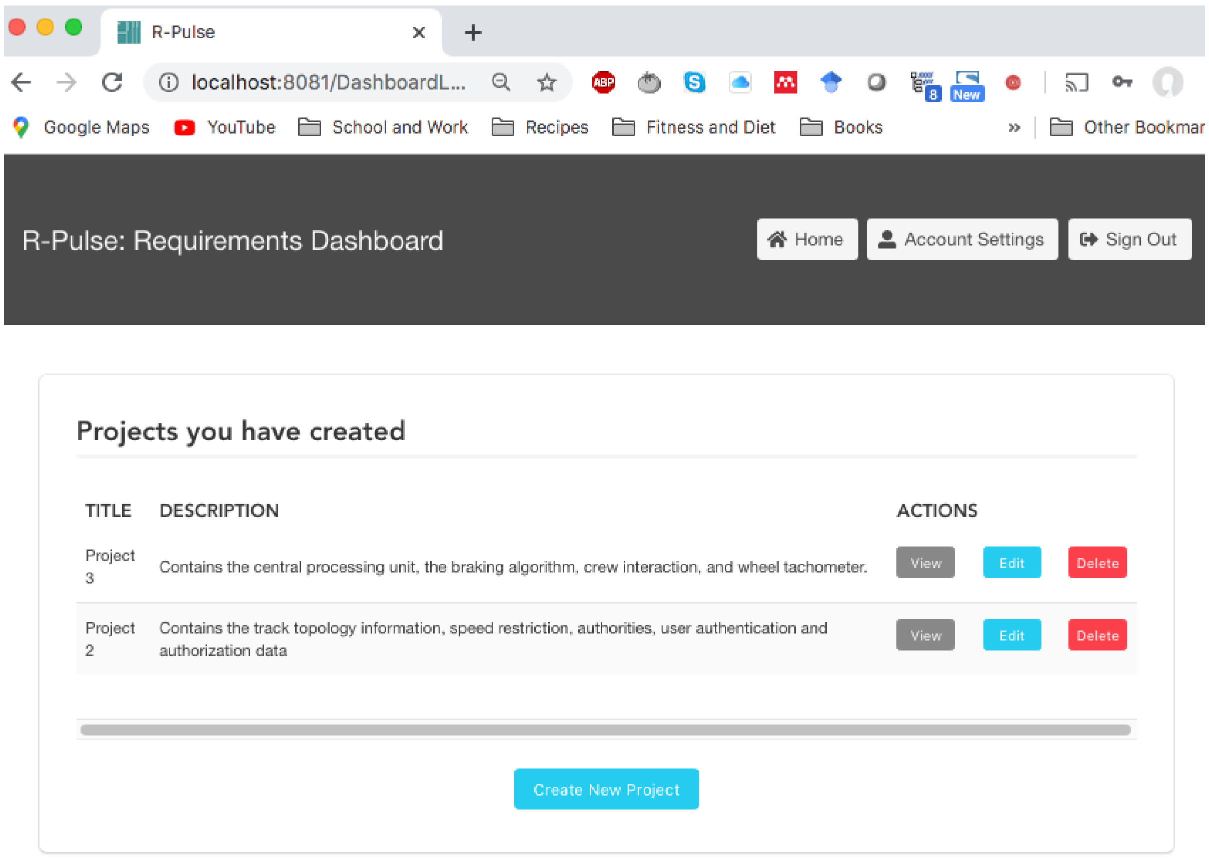
Levels List

TITLE
Baseline
Feature
Release
Safety

Figure 7.3: Edit artifact page in R-Pulse.

Data Analysis The data analysis functionality in the dashboard performs the following two main functions: calculating the requirements measures according to the metrics defined in Chapter 3 and calculating the health indicators according to the analysis models described in Chapter 5 using the data that was entered and uploaded into the dashboard.

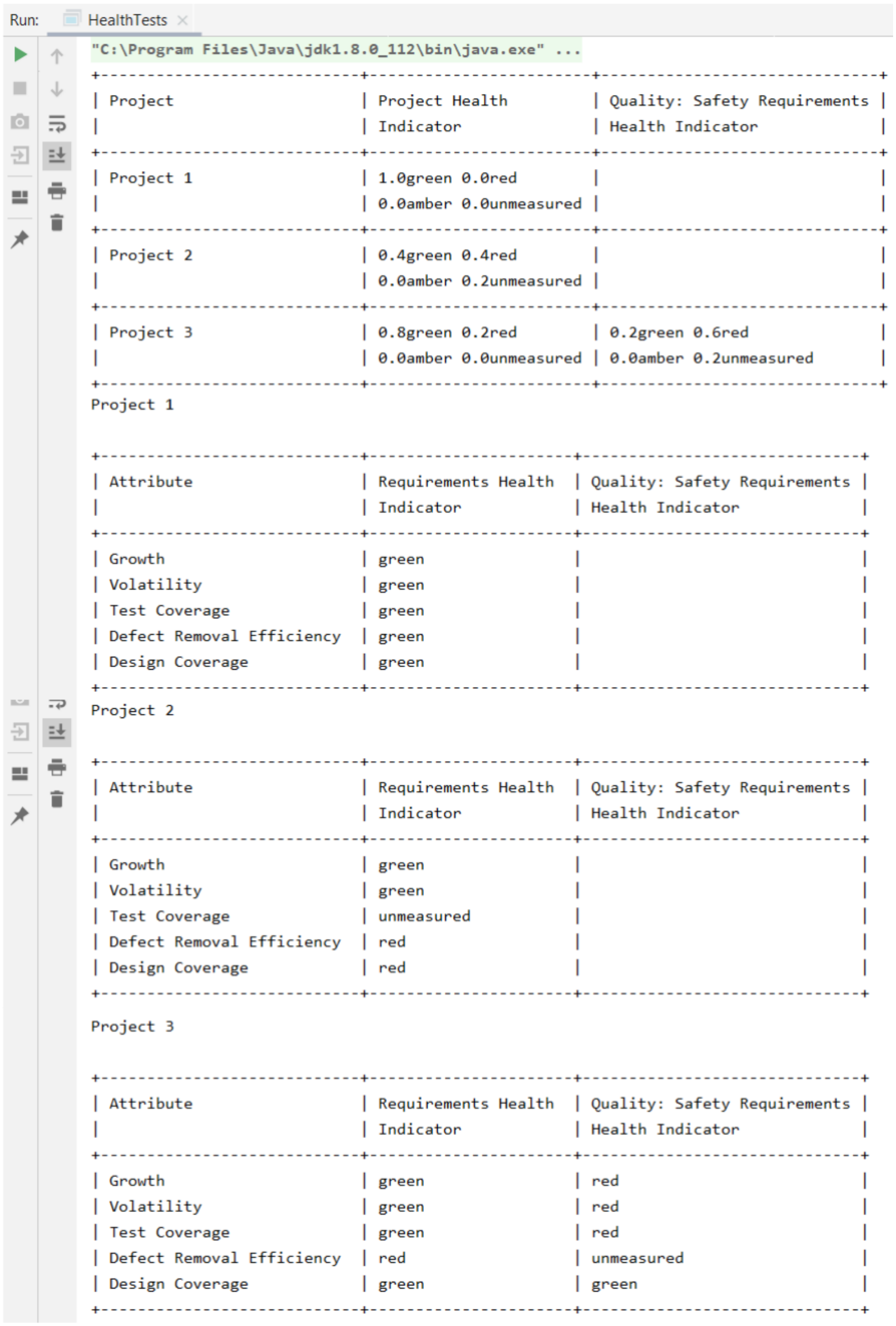
Data Visualization and Navigation The dashboard provides a range of data visualization and navigation options to the internal stakeholders. For example, a program manager can navigate between different project data as shown in Figure 7.4. Each project's health indicators are displayed according to concepts described in Chapter 5. While the back-end code for the health indicators has been implemented as shown in Figure 7.5, the front end is currently under development.



Projects you have created

TITLE	DESCRIPTION	ACTIONS		
Project 3	Contains the central processing unit, the braking algorithm, crew interaction, and wheel tachometer.	View	Edit	Delete
Project 2	Contains the track topology information, speed restriction, authorities, user authentication and authorization data	View	Edit	Delete

Figure 7.4: Navigate projects in R-Pulse.

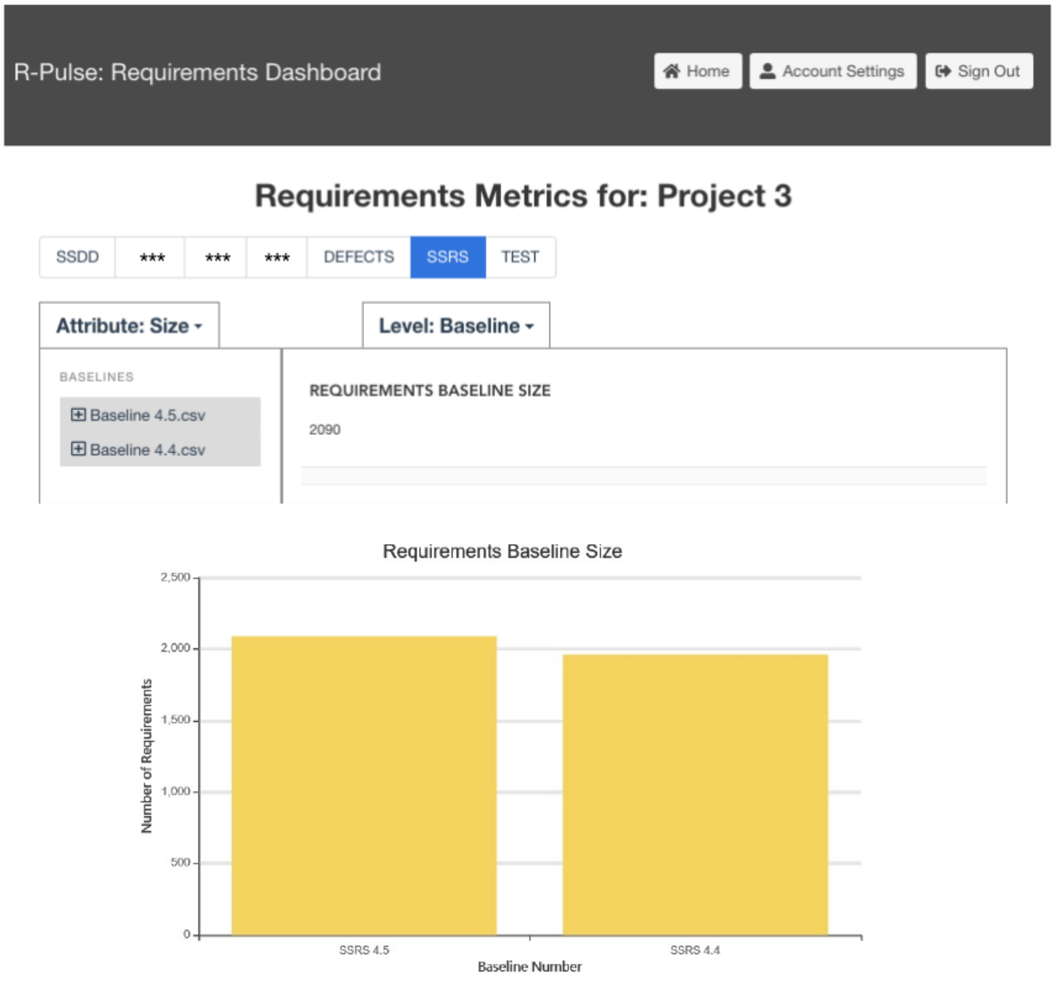


```
"C:\Program Files\Java\jdk1.8.0_112\bin\java.exe" ...
+----------------------------+-----------------------+------------------------------+
| Project                    | Project Health        | Quality: Safety Requirements |
|                            | Indicator             | Health Indicator             |
+----------------------------+-----------------------+------------------------------+
| Project 1                  | 1.0green 0.0red       |                              |
|                            | 0.0amber 0.0unmeasured |                             |
+----------------------------+-----------------------+------------------------------+
| Project 2                  | 0.4green 0.4red       |                              |
|                            | 0.0amber 0.2unmeasured |                             |
+----------------------------+-----------------------+------------------------------+
| Project 3                  | 0.8green 0.2red       | 0.2green 0.6red              |
|                            | 0.0amber 0.0unmeasured | 0.0amber 0.2unmeasured      |
+----------------------------+-----------------------+------------------------------+
Project 1

+----------------------------+----------------------+------------------------------+
| Attribute                  | Requirements Health  | Quality: Safety Requirements |
|                            | Indicator            | Health Indicator             |
+----------------------------+----------------------+------------------------------+
| Growth                     | green                |                              |
| Volatility                 | green                |                              |
| Test Coverage              | green                |                              |
| Defect Removal Efficiency  | green                |                              |
| Design Coverage            | green                |                              |
+----------------------------+----------------------+------------------------------+
Project 2

+----------------------------+----------------------+------------------------------+
| Attribute                  | Requirements Health  | Quality: Safety Requirements |
|                            | Indicator            | Health Indicator             |
+----------------------------+----------------------+------------------------------+
| Growth                     | green                |                              |
| Volatility                 | green                |                              |
| Test Coverage              | unmeasured           |                              |
| Defect Removal Efficiency  | red                  |                              |
| Design Coverage            | red                  |                              |
+----------------------------+----------------------+------------------------------+

Project 3

+----------------------------+----------------------+------------------------------+
| Attribute                  | Requirements Health  | Quality: Safety Requirements |
|                            | Indicator            | Health Indicator             |
+----------------------------+----------------------+------------------------------+
| Growth                     | green                | red                          |
| Volatility                 | green                | red                          |
| Test Coverage              | green                | red                          |
| Defect Removal Efficiency  | red                  | unmeasured                   |
| Design Coverage            | green                | green                        |
+----------------------------+----------------------+------------------------------+
```

Figure 7.5: The back-end implementation of the requirements-centric health indicators.

Internal stakeholders can view each artifact's raw measures as shown in Figure 7.6. Further navigation between an artifact's *attributes* and levels is possible as shown in Figures 7.7 and 7.8. In addition to displaying the measures in tables, the measures are visualized in various graphs and charts that the internal stakeholder can choose from such as line graphs, pie charts, and bar charts as seen in Figure 7.6.

We note that thus far we have been testing the dashboard with small portions of the projects' data. We have yet to test the dashboard fully on complete and correct project data.



Requirements Metrics for: Project 3

REQUIREMENTS BASELINE SIZE

2090

Figure 7.6: Viewing each project's requirements metrics.
The horizontal menu at the top indicates the project's artifacts that consists of the SSDD (*Subsystem Design Document*), and SSRS (*Subsystem Requirements Specification*), defects documents, and test cases. The obfuscated artifacts are proprietary artifacts and have been obfuscated due to confidentiality reasons.

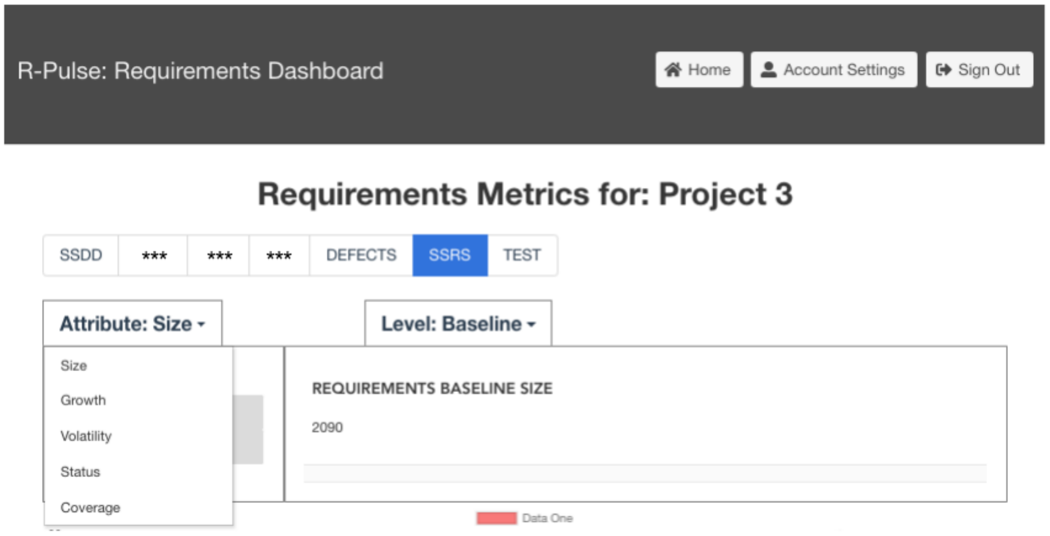


Figure 7.7: Filtering of metrics according to requirements attributes.

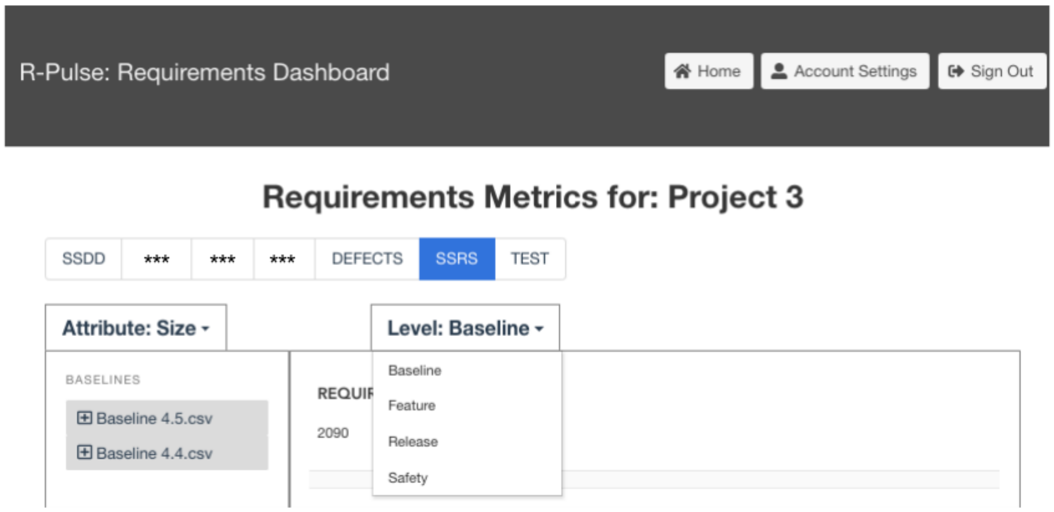


Figure 7.8: Filtering of metrics according to requirements levels.

7.3.2 Architecture

Figure 7.9 depicts the dashboard's three-tier architecture which consists of data, application, and presentation layers. Such an architecture is recommended for measurement tools as it allows for separating the different functionalities of the measurement process [Staron and Meding, 2018].

Data Layer The data layer consists of the database in which the input data is stored. We used Maria DB for building the dashboard's database. As depicted in Figure 7.9, the collected data is organized into the following tables in the database: project, attribute, level, artifact, user, role, and a number of connector tables.

Application Layer The data analysis functionality of the dashboard takes place in the application layer which is implemented in Java and uses an R package to run the statistical analyses required for the health indicators. Specifically, the formulas for calculating the measures are written in Java in *formulas* and *metrics* packages. The resulting measures are then used as input into the *healthindicator* package.

Presentation Layer The presentation layer is a web application that consumes data from the data layer. Bootstrap and Vue.js were used to implement the visualization and navigation features discussed above.

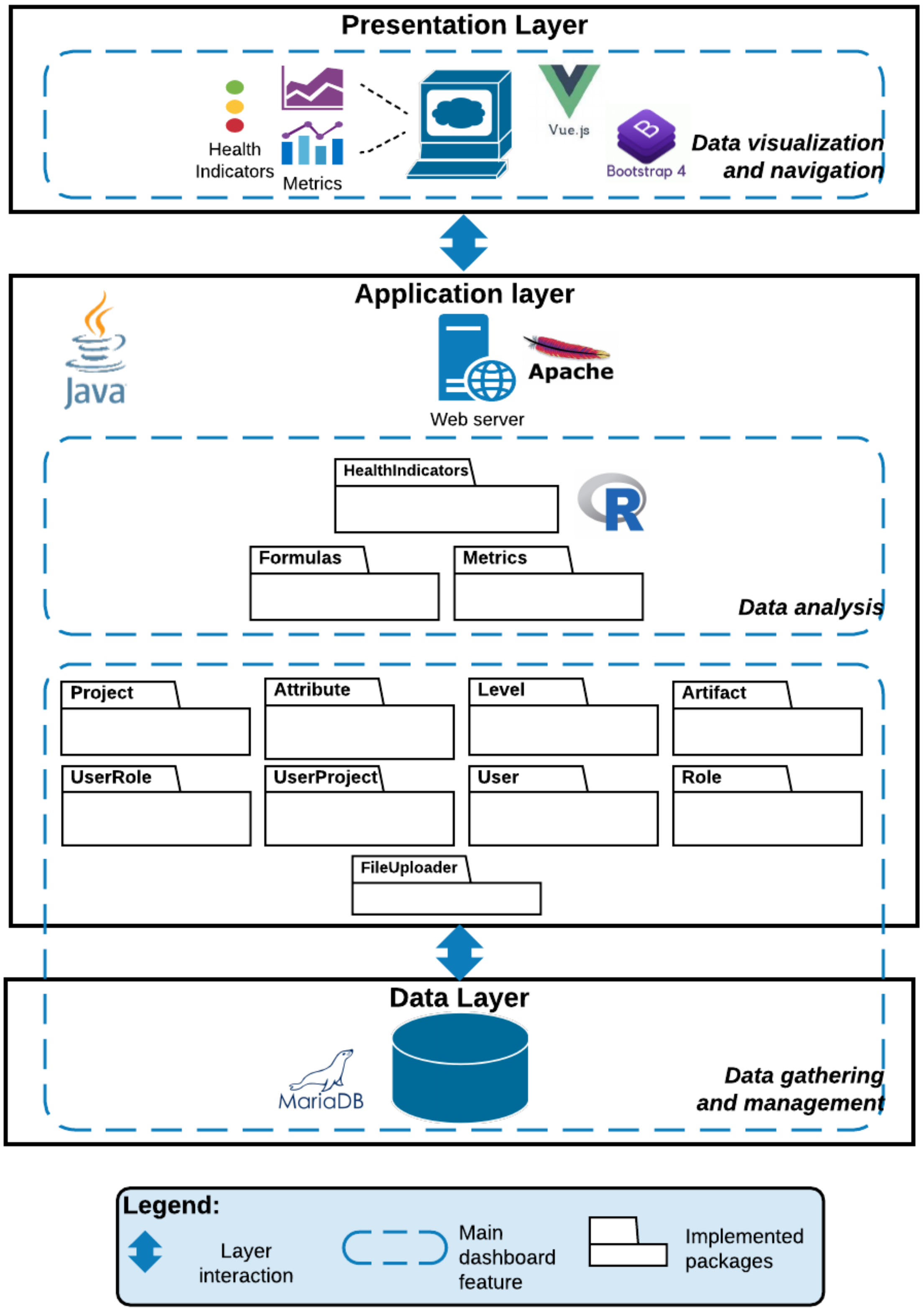


Figure 7.9: Dashboard architecture.

7.4 Dashboard Development Process

An iterative development approach [Larman and Basili, 2003] was be used to build the dashboard prototype. Iterative design and development methodologies involve a cyclic process of requirements planning, design and analysis, implementation, testing, and evaluation as depicted in Figure 7.10, which we describe below.

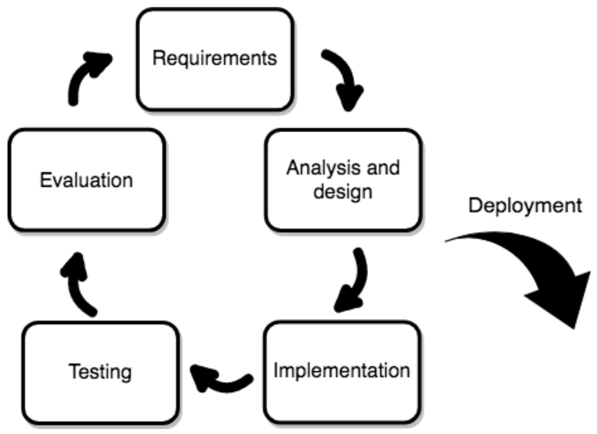


Figure 7.10: Dashboard development process.

Requirements The requirements for the dashboard were elicited on-site during the AR study (see Chapter 3 for AR study details) over a period of 10 months from February 2017 to December 2017. The requirements were then analyzed and refined after the AR study in consultation with the stakeholders within the projects iteratively over the course of the dashboard development process. The concepts underlying the dashboard (i.e., metrics, attributes, levels, internal stakeholder categories, and health indicators) were communicated and explained to the development team.

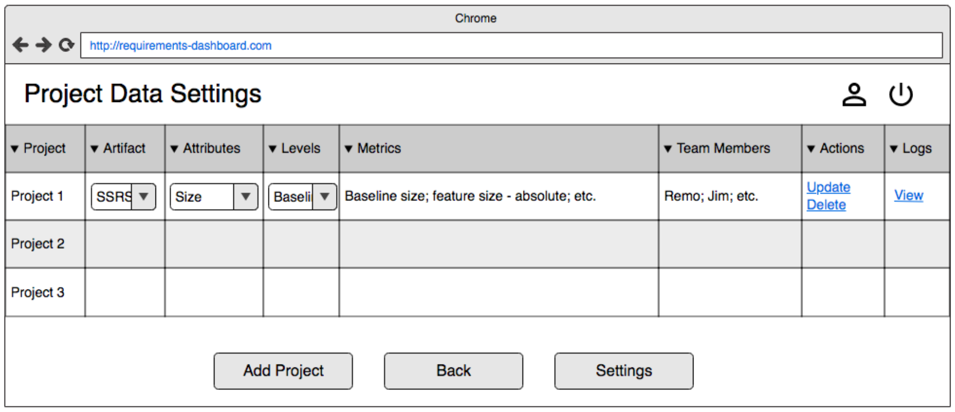
Analysis and Design The analysis and design stage took place between September 2018 to August 2019 and consisted of the following activities:

1. Creating mock-ups of the dashboard's graphical user interface using an online webtool (moqups.com). Figures 7.11 and 7.12 show samples of the dashboard mock ups.

2. Creating the dashboard's class diagram.

3. Sketching the system's architecture.

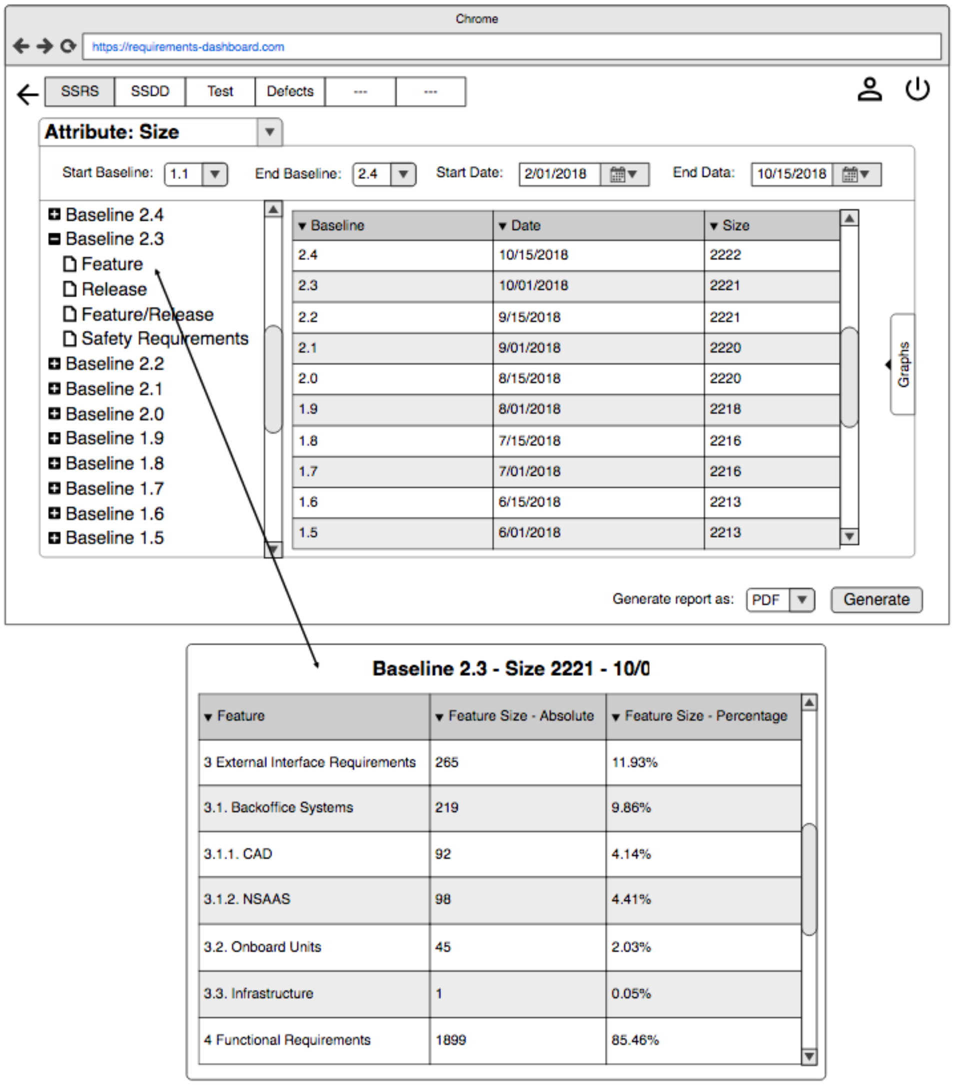
4. Choosing the tools to be used to build the dashboard such as back-end and front-end programming languages, database, and so on.



Project Data Settings

▾ Project	▾ Artifact	▾ Attributes	▾ Levels	▾ Metrics	▾ Team Members	▾ Actions	▾ Logs
Project 1	SSRS	Size	Baseli	Baseline size; feature size - absolute; etc.	Remo; Jim; etc.	Update Delete	View
Project 2							
Project 3							

Figure 7.11: Dashboard mock-up for the *Projects* page.



Attribute: Size

Start Baseline: 1.1 End Baseline: 2.4 Start Date: 2/01/2018 End Data: 10/15/2018

- Baseline 2.4
- Baseline 2.3
  - Feature
  - Release
  - Feature/Release
  - Safety Requirements
- Baseline 2.2
- Baseline 2.1
- Baseline 2.0
- Baseline 1.9
- Baseline 1.8
- Baseline 1.7
- Baseline 1.6
- Baseline 1.5

▾ Baseline	▾ Date	▾ Size
2.4	10/15/2018	2222
2.3	10/01/2018	2221
2.2	9/15/2018	2221
2.1	9/01/2018	2220
2.0	8/15/2018	2220
1.9	8/01/2018	2218
1.8	7/15/2018	2216
1.7	7/01/2018	2216
1.6	6/15/2018	2213
1.5	6/01/2018	2213

Generate report as: PDF

Baseline 2.3 - Size 2221 - 10/0

▾ Feature	▾ Feature Size - Absolute	▾ Feature Size - Percentage
3 External Interface Requirements	265	11.93%
3.1. Backoffice Systems	219	9.86%
3.1.1. CAD	92	4.14%
3.1.2. NSAAS	98	4.41%
3.2. Onboard Units	45	2.03%
3.3. Infrastructure	1	0.05%
4 Functional Requirements	1899	85.46%

Figure 7.12: Dashboard mock-up for the *Requirements Size Metrics* page.

Implementation The implementation of the dashboard began in January 2019 and is still ongoing. The implementation phase consisted of the database creation and back-end and front-end development.

Testing We performed some testing during the development process of the dashboard and further testing will be carried out after completing the dashboard development. The following testing methods have been used or will be used as the development of R-Pulse progresses:

1. **Unit testing:** A combination of manual and automated unit testing was performed during the development process to test the dashboard's functions and objects such as data entering, data retrieval, and metric calculations. We performed unit testing as we developed the dashboard using mostly mock data.

2. **Integration testing:** Integration testing was conducted between the dashboard's different objects such as the metrics object and the health indicators object and communication between back-end and front-end of the dashboard. Both unit and integration testing was performed by the developers.

3. **System testing:** We performed some system testing on the functional parts of the dashboard. System testing was carried out by individuals separate from the development team (e.g., external junior and senior researchers).

4. **Acceptance testing:** We intend to perform acceptance testing through the internal stakeholders of the collaborating organization.

5. **Usability testing:** Usability testing will be performed upon deploying the dashboard within the projects in the collaborating organization.

Evaluation The evaluation of the dashboard consists of evaluating it for *usability* and *relevance*. To evaluate its usability, a form of usability testing will be performed as discussed in the preceding paragraph. To evaluate its relevance to requirements management, measurement, and process development in general, we intend to evaluate it through practitioners outside of the collaborating organization and in other domains. Given that the dashboard is thus far a working prototype, an extensive evaluation of its usability and relevance has yet to be conducted. However, we acquired preliminary evidence of its usability through the following methods.

First, the dashboard's requirements have been elicited based on the organization's internal stakeholder information needs and their requirements and development processes. The researcher was immersed onsite and actively involved in the projects as requirements engineer for a period of 10 months (see Chapter 2). Thus, the dashboard requirements emerged from the industrial setting (see Section 7.2).

Second, we designed R-Pulse, thus far, with continuous feedback from the internal stakeholders at the collaborating organization. Specifically, we conducted several interviews with the main requirements internal stakeholder in the organization to evaluate the the mock-up designs and ensure that they met the internal stakeholder needs. The feedback we acquired from the internal stakeholder was then incorporated into the dashboard design. The feedback concerned organization of pages and data and additional functionality.

Third, four external assessors, two researchers and two practitioners, are currently using the dashboard prototype to provide feedback on its usability.

7.5 Related Work and Comparison

ISEMA is an approach to software measurement that has its roots since the late 1980s [Basili and Rombach, 1988] and early 1990s [Selby et al., 1991]. Particularly, software measurement dashboards began gaining prominence in the early 2000s and have continued to this day. Among the early dashboards are the Software Process Dashboard [Tuma, 2000], Pro Metrics (PROM) [Sillitti et al., 2003], Hackystat [Johnson et al., 2003; Johnson, 2007], and Empirical Project Monitor (EPM) [Ohira et al., 2004]. More recently, dashboards and measurement tools such as Project Insights and Visualization Toolkit (PIVot) [Sharma and Kaulgud, 2012], and Quality-aware Rapid Software Development (Q-Rapids) tool [López et al., 2018] have been proposed. We describe each of the dashboards below and compare it to R-Pulse.

The Software Process Dashboard is a tool that automates the techniques defined by the Personal Software Process (PSP) [Humphrey, 1995] that adapts organizational-level software measurement and analysis techniques to the individual developer. The dashboard was originally developed in 1998 by the United States Air Force, and has continued to evolve under the open-source model. The Process Dashboard supports data (e.g., time, code defects, and code size) collection, planning (e.g., templates, forms, and summaries), tracking (i.e., earned value support), data analysis (e.g., charts and reports), and data exporting into Excel or text format for use with external tools.

PROM [Sillitti et al., 2003] is tool for automated data gathering and analysis that calculates both code and process measures, including PSP metrics, procedural and and object oriented metrics. PROM collects and analyzes data at different levels of granularity: personal, team and enterprise. This distinction is intended to preserve developers' privacy while providing aggregated, anonymous data to managers.

Hackystat [Johnson et al., 2003; Johnson, 2007] is one of the more popular software measurement tools that automatically collects individual development data through sensors attached to development tools and IDEs (e.g., Eclipse, EMACS, JBuilder, and Vim), testing tools (e.g., JUnit and CppUnit), and static analysis tools (e.g., CheckStyle, FindBugs, and PMD). The sensors communicate with a centralized server using SOAP protocols. Developer behaviour and measures are then displayed on a web page for each individual developer.

EPM [Ohira et al., 2004] automatically collects and measures quantitative data from three kinds of repositories: versioning histories from configuration management systems (e.g. CVS), mail archives from mailing list managers (e.g. Mailman, Majordomo, and fml), and issue tracking records from (bug) issue tracking systems (e.g. GNATS5 and Bugzilla). EPM consists of four components: data collection, format translation, data store, and data analysis/visualization. EPM visualizes the various measures (e.g., growth of lines of code and relationship between check-in and checkout) and provides summaries of each repository. Measures and visualizations can be accessed using common web browsers.

PIVoT [Sharma and Kaulgud, 2012] is a tool that provides project managers with a holistic picture of a project's health and trajectory. PIVoT collects and visualizes thirty software metrics related to code quality, quality of component, testing effort, development efficiency, code churn, and team analysis. The measures are calculated and visualized in the tool through an assortment of graphs.

Finally, the Q-Rapids method and tool [López et al., 2018] is part of larger project that gathers data from different software and project repositories (e.g., Jira and SonarQube), analyzes the gathered data, and presents quality-related data to decision-makers using a dashboard. The current release of Q-Rapids Tool provides four sets of functionality: (1) data gathering from source tools (e.g. GitLab, Jira, SonarQube, and Jenkins), (2) aggregation of data into three levels of abstraction: metrics (e.g., code bug density, availability, and uptime), product/process factors (e.g., code quality, and software usage), and strategic indicators (e.g., customer satisfaction, and product quality), (3) visualization of the aggregated data, (4) and navigation of the aggregated data.

On a more generic level, Tableau [Chabot et al., 2003] is a commercial tool that extracts data from a variety of sources and provides users with data analysis options and the results are visualized in dashboards. Tableau's strength lies in its ability to connect from a large number of platforms including spreadsheets, PDF files, relational databases, cloud-based databases such as Amazon webs services, Microsoft Azure SQL database, and Google Cloud SQL. Moreover, it allows for advanced data visualization and interaction features.

The above survey of software measurement dashboards shows that there is no *one-size-fits-all* dashboard that supports all software development needs. The need for different dashboards for different industries, sub-industries, and processes has been recognized in the research community [Pauwels et al., 2009]. Thus, R-Pulse adds to the work on software measurement dashboards by treating requirements as the main measured entity. As we've seen in the review above, the majority of software dashboards are based on code-centric metrics with little to no consideration for requirements. Moreover, the dashboards are heavily biased towards developers with the exception of PIVot and the Q-Rapids tool. The levels of abstraction displayed in R-Pulse (i.e., health indicators, absolute measures, percentage, and visualizations) cater to a wider range of users. While the Q-rapids tool comes closest to our work, the measures remain code-centric, which are then aggregated into quality attributes. Thus, we can consider the data and functionality in Q-Rapids and R-Pulse as complementary; product quality measures and indicators from Q-Rapids can be used in conjunction with requirement metrics to provide more accurate project health indicators (see Chapter 5) while requirements measures from R-Pulse can be used with code measures in Q-Rapids to better assess product quality.

Finally, while commercial software such as Tableau has the power to connect to any data source, one would still require the concepts in R-pulse (i.e., metrics and health indicators) to analyze the data retrieved from the data sources in Tableau to create custom dashboards. Moreover, such commercial tools are expensive which is an added cost that not all companies and/or projects would be willing to pay.

7.6 Limitations

As with any solution, R-Pulse has its limitations which we discuss in this section.

The first limitation is the dashboard's usability in a context different than the large systems engineering context it was built for. More specifically, R-Pulse was built for large systems engineering projects that have established RE processes, advanced requirements documentation procedures, and numerous internal stakeholders with diverse requirements-related concerns. Thus, its usability in other contexts, such as agile processes, that have limited support for requirements documentation and small teams may be limited as well.

The second limitation is the dashboard's reliance on spreadsheets and not being directly connected to the requirements database. This can cause several problems. For one thing, the exporting of requirements and associated meta-data and uploading it to the dashboard is error-prone and may introduce inconsistencies and incompletenesses in the imported data. Moreover, the exporting of baselines from the requirements databases and importing it into the dashboard is an added effort that the projects may not be willing to expend.

The third limitation is the minimal evaluation we conducted thus far of the dashboard's usability and ease of use in practice as full deployment in the organization is still underway.

7.7 Conclusions and Future Work

Tool support is essential for the facilitation of measurement in systems and software processes and is key for the success of a measurement program. Particularly, requirements measurement presents its own set of challenges when done manually or semi-manually (Section 7.2). Current measurement tools and dashboards focus largely on code-centric measurement (Section 7.5) with little support for requirements measurement and requirements management tools also lack support for requirements measurement.

In this paper, we present R-Pulse, a web-based requirements measurement dashboard that has the following main features: data gathering and management, data analysis, and data visualization and navigation (Section 7.3). We built R-Pulse in collaboration with an organization in the systems engineering domain and adopted a 3-tier architecture.

R-Pulse is still a prototype and much remains to be done with regard to its development and evaluation. Thus, we delineate here several short-term and long-term action items. The most imminent task at the current stage of development is carrying out the acceptance and usability tests in order to evaluate its usability in a real-life setting. Specifically, we plan to deploy the dashboard within the collaborating organization and evaluate its usability. Next, we plan to evaluate the dashboard's relevance and generalizability to other contexts through deploying it in different organizations and gathering further feedback on its use and relevance to practitioners.

The long-term action items include developing more advanced features for the dashboard such as requirements-centric prediction of defects, cost, and effort and advanced analytics that are based on requirement measurement. Moreover, the dashboard may benefit from a micro-services architecture as it will allow easier extendability of the dashboard functionality.

References

[Basili and Rombach, 1988] Basili, V. R. and Rombach, D. H. (1988). The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773.

[Chabot et al., 2003] Chabot, C., Hanrahan, P., and Stolte, C. (2003). Tableau. Available at https://www.tableau.com/.

[Coman et al., 2009] Coman, I. D., Sillitti, A., and Succi, G. (2009). A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*, pages 89–99, Vancouver, Canada. IEEE.

[Daskalantonakis, 1992] Daskalantonakis, M. K. (1992). A Practical view of software measurement and implementation experiences within Motorola. *IEEE Transactions on Software Engineering*, 18(11):998–1010.

[Ebert and Dumke, 2007] Ebert, C. and Dumke, R. (2007). *Software measurement: establish, extract, evaluate, execute*. Springer.

[Gopal et al., 2002] Gopal, A., Krishnan, M. S., Mukhopadhyay, T., and Goldenson, D. R. (2002). Measurement programs in software development: Determinants of success. *IEEE Transactions on Software Engineering*, 28(9):863–875.

[Hall and Fenton, 1997] Hall, T. and Fenton, N. (1997). Implementing effective software metrics programs. *IEEE Software*, 14(2):55–64.

[Humphrey, 1995] Humphrey, W. S. (1995). *A discipline for software engineering*. Addison-Wesley.

[Iversen and Mathiassen, 2000] Iversen, J. and Mathiassen, L. (2000). Lessons from implementing a software metrics program. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS'00)*, Maui, HI, USA. IEEE.

[Johnson, 2007] Johnson, P. M. (2007). Requirement and design tradeoffs in Hackystat: An inprocess software engineering measurement and analysis system. In *1st International Symposium on Empirical Software Engineering and Measurement*, pages 81–90, Madrid, Spain. IEEE.

[Johnson et al., 2003] Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Zhen, S., and Doane, W. E. (2003). Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the International Conference on Software Engineering (ICSE'03)*, volume 6, pages 641–646, Portland, OR. IEEE.

[Kerzner, 2017] Kerzner, H. (2017). *Project management metrics, KPIs, and dashboards: A guide to measuring and monitoring project performance*. John and Wiley & Sons, third edition.

[Larman and Basili, 2003] Larman, C. and Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, 36(6):47–56.

[López et al., 2018] López, L., Martínez-Fernández, S., Gómez, C., Choraś, M., Kozik, R., Guzmán, L., Vollmer, A. M., Franch, X., and Jedlitschka, A. (2018). Q-rapids tool prototype: Supporting decision-makers in managing quality in rapid software development. In *Lecture Notes in Business Information Processing*, volume 317, pages 200–208.

[Ohira et al., 2004] Ohira, M., Yokomori, R., and Sakai, M. (2004). Empirical project monitor: A tool for mining multiple project data. In *Proceedings of the International Workshop on Mining Software Repositories (MSR'04)*, pages 42–46, Edinburgh, UK.

[Pauwels et al., 2009] Pauwels, K., Ambler, T., Clark, B. H., LaPointe, P., Reibstein, D., Skiera, B., Wierenga, B., and Wiesel, T. (2009). Dashboards as a service. *Journal of Service Research*, 12(2):175–189.

[Pfleeger, 1993] Pfleeger, S. L. (1993). Lessons learned in building a corporate metrics program. *IEEE Software*, 10(3):67–74.

[Selby et al., 1991] Selby, R., Porter, A., Schmidt, D., and Berney, J. (1991). Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of the 13th International Conference on Software Engineering (ICSE'91)*, pages 288–298, Austin, TX, USA. IEEE.

[Selby, 2009] Selby, R. W. (2009). Analytics-driven dashboards enable leading indicators for requirements and designs of large-scale systems. *IEEE Software*, 26(1):41–49.

[Sharma and Kaulgud, 2012] Sharma, V. S. and Kaulgud, V. (2012). PIVoT: Project insights and visualization toolkit. In Zurich, S., editor, *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM'12)*, pages 63–69. IEEE.

[Sillitti et al., 2003] Sillitti, A., Janes, A., Succi, G., and Vernazza, T. (2003). Collecting, integrating and analyzing software metrics and personal software process data. In *Proceedings of the 29th EUROMICRO Conference*, pages 336–342, Belek-Antalya, Turkey. IEEE Computer Society.

[Staron and Meding, 2018] Staron, M. and Meding, W. (2018). *Software development measurement programs: Development, management and evolution.* Springer.

[Tuma, 2000] Tuma, S. (2000). The software process dashboard initiative. Available at https://www.processdash.com/home.

Chapter 8

Discussion

Each of the previous chapters (3, 4, 5, 6, and 7) constituted a component of the requirements measurement program that this thesis aims to construct (see Chapter 1 for research goal and objectives). Figure 8.1 provides an overview of the measurement program components (MPC) presented in this thesis.

This chapter has three objectives: 1) to situate each measurement program component within established measurement process standards [IEEE, 2017] (i.e., research) and within the collaborating organization's requirements engineering process (i.e., practice) (Section 8.1), 2) to revisit and summarize the validation of each of the requirements measurement program components (Section 8.2), and 3) to evaluate the overall robustness of the measurement program according to established evaluation criteria [Staron and Meding, 2018] (Section 8.3).

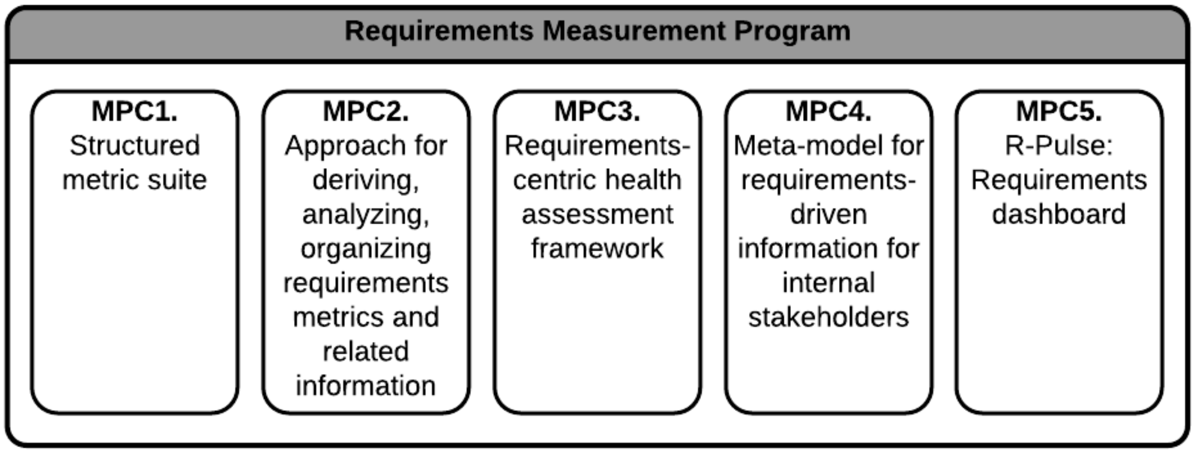

Requirements Measurement Program
MPC1. Structured metric suite
MPC2. Approach for deriving, analyzing, organizing requirements metrics and related information
MPC3. Requirements-centric health assessment framework
MPC4. Meta-model for requirements-driven information for internal stakeholders
MPC5. R-Pulse: Requirements dashboard

Figure 8.1: Requirements measurement program components presented in thesis.

8.1 Situating the Requirements Measurement Program

Figure 8.2 depicts the general measurement process as established by [IEEE, 2017] and situates each of the measurement program components within that process. Particularly, Figure 8.2 shows which measurement task each measurement program component supports. We can see that the various measurement program components mainly support the tasks within the *prepare for measurement* and *perform measurement* phases of the measurement process. Both phases are considered the cornerstone of the measurement process and require the most time, resources, and effort.

The entirety of the measurement process revolves around selecting and specifying metrics that satisfy the information needs of the project or organization (task 2.4 in Figure 8.2). The *structured requirements metrics suite* (Chapter 3) supports this task within the context of requirements engineering by providing a set of empirically derived, well-defined, and validated set of requirements metrics that we have shown to address different internal stakeholders' needs.

However, every project or organization has different information needs with regard to requirements measures. Moreover, the same project's information needs may evolve over time. Thus, the *approach for deriving, analyzing, and organizing metrics and related information* (Chapter 4) also supports the task of selecting and specifying metrics by providing projects and organizations with a means to derive further metrics that address their evolving needs. In addition, as described in Chapter 4, the approach also can aid in analyzing metrics and organizing metrics and measurement reports. It, then, also aids in the task of defining collection, analysis, and and reporting procedures (task 2.5 in Figure 8.2).

The *requirements-centric health assessment framework* (Chapter 5) supports the task of defining analysis procedures in the *prepare* phase (task 2.5 in Figure 8.2) through the analysis models of the framework that analyzes requirements measures in conjunction with project data. The result of the analysis models in the framework, which is in the form of red-amber-green (RAG) health indicators, supports the task of developing the information items to be disseminated to the internal stakeholders (task 3.3 in Figure 8.2).

The entities and relationships at abstraction levels 1 and 2 in the *meta-model for requirements-driven information for internal stakeholders* (Chapter 6) can guide the definition of the requirements measurement strategy (task 2.1 in Figure 8.2) within a project. In addition, the meta-model entities at abstract levels 2 and 3 can aid in describing the organization characteristics and entities relevant to the requirements measurement procedure (task 2.2).

Finally, the *requirements dashboard* (Chapter 7) assists the *perform measurement* phase of the measurement process by automating all the measurement tasks in the phase.

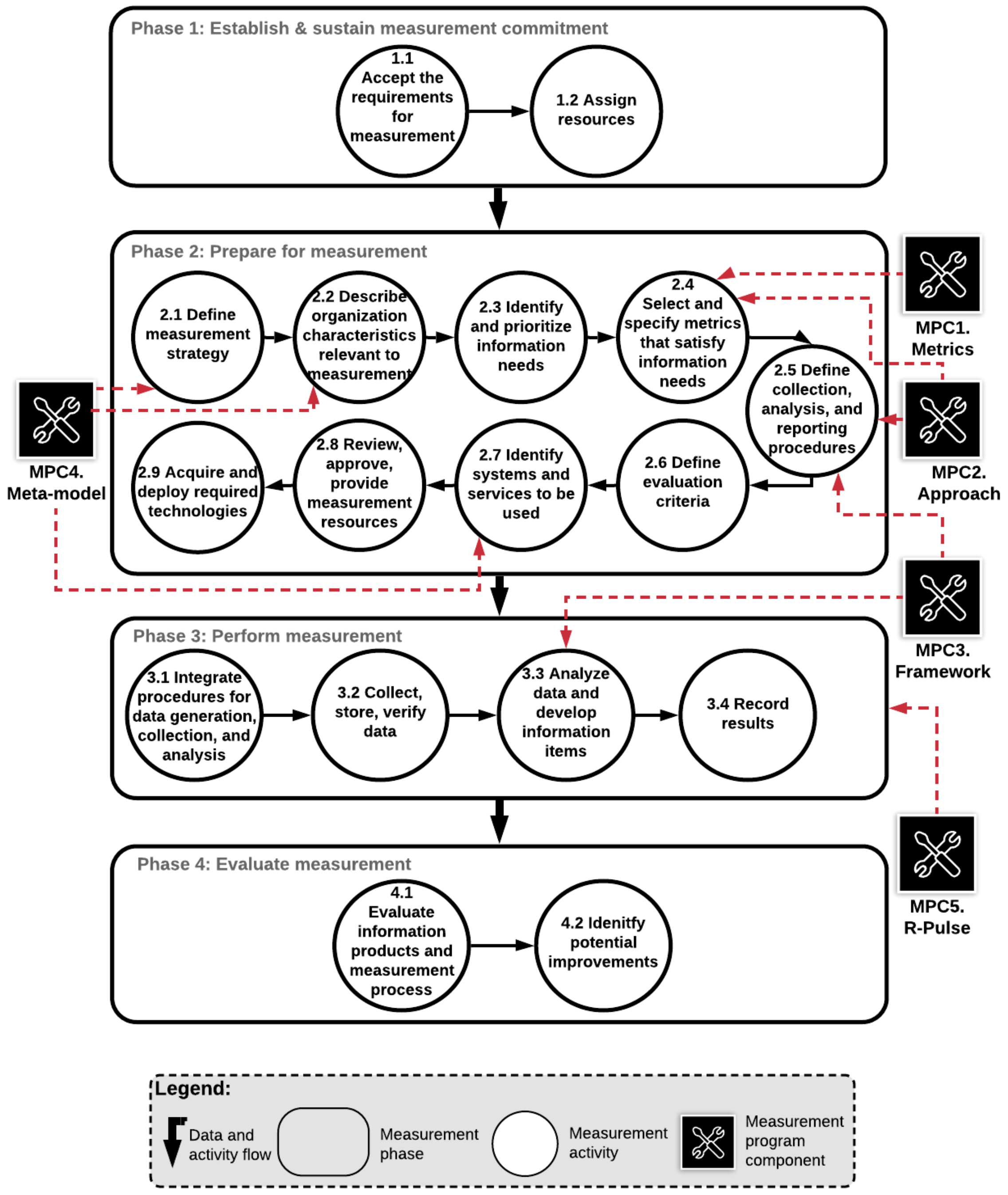

Phase 1: Establish & sustain measurement commitment
1.1 Accept the requirements for measurement
1.2 Assign resources
Phase 2: Prepare for measurement
2.1 Define measurement strategy
2.2 Describe organization characteristics relevant to measurement
2.3 Identify and prioritize information needs
2.4 Select and specify metrics that satisfy information needs
2.5 Define collection, analysis, and reporting procedures
2.6 Define evaluation criteria
2.7 Identify systems and services to be used
2.8 Review, approve, provide measurement resources
2.9 Acquire and deploy required technologies
Phase 3: Perform measurement
3.1 Integrate procedures for data generation, collection, and analysis
3.2 Collect, store, verify data
3.3 Analyze data and develop information items
3.4 Record results
Phase 4: Evaluate measurement
4.1 Evaluate information products and measurement process
4.2 Idenitfy potential improvements
MPC1. Metrics
MPC2. Approach
MPC3. Framework
MPC4. Meta-model
MPC5. R-Pulse
Legend:
Data and activity flow
Measurement phase
Measurement activity
Measurement program component

Figure 8.2: Measurement program components situated in the measurement process.

In Chapter 2 we described the industrial context from which this work emerged. Particularly, we detailed their requirements engineering and management processes. While requirements measurement is enlisted as one of the activities of the requirements management process in theory, no requirements metrics, approaches and tools were used or defined before the commencement of this study. Figure 8.3 depicts the RE process in the organization and situates the requirements measurement process within it, which utilizes the measurement program components presented in this thesis.

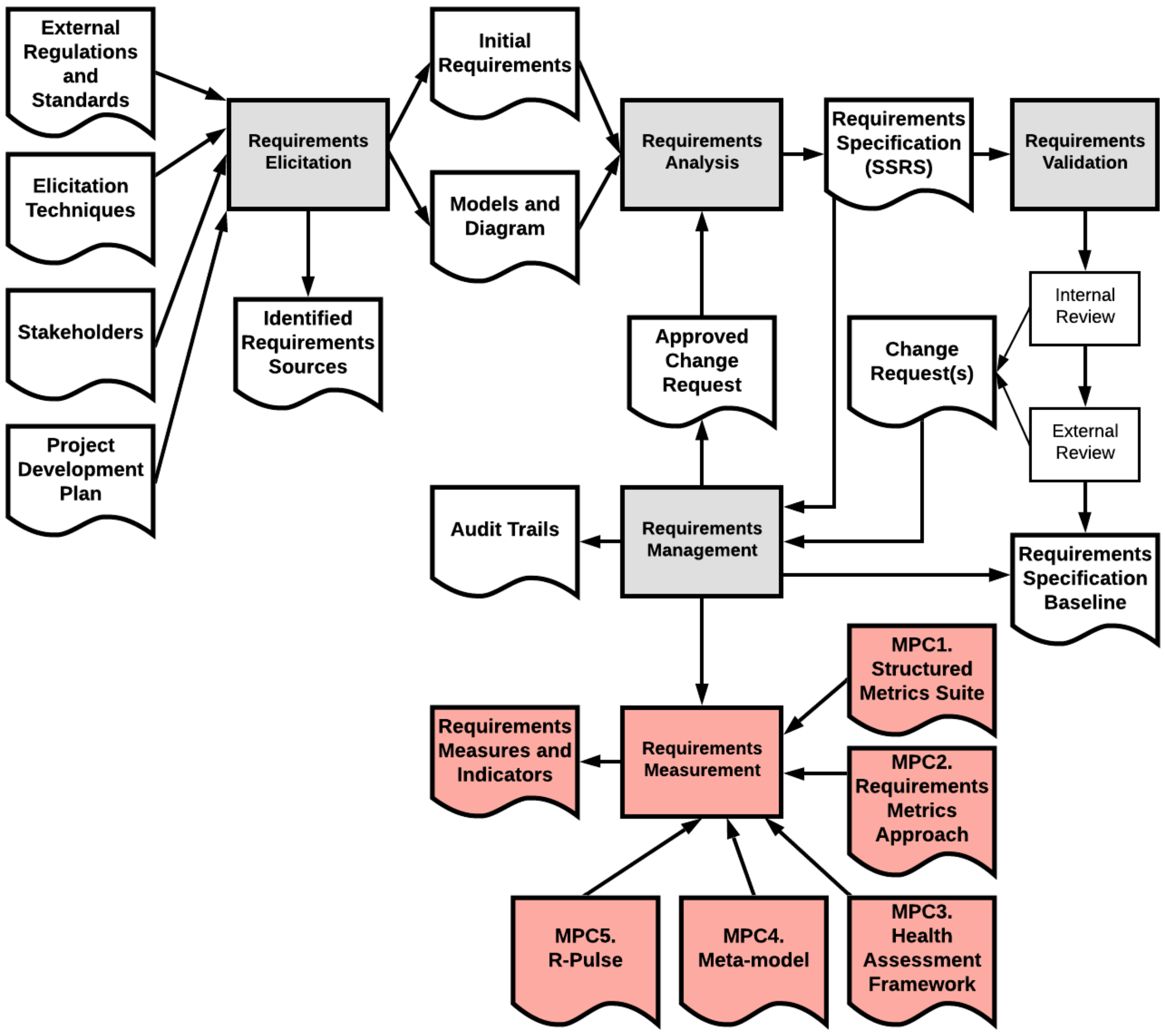


Figure 8.3: Measurement program within the RE process in the collaborating organization.

8.2 Summary of Validation of Requirements Measurement Program Components

Before we evaluate the overall measurement program in the following section (Section 8.3), we summarize the validation conducted for each component of the measurement program in Table 8.1. The reader can refer to the measurement program components' corresponding chapters for details on the validation procedures and results.

Table 8.1: Summary of validation of requirements measurement program components.

Thesis Chapter	**Measurement Program Component (MPC)**	**Theoretical Validation**	**Empirical Validation**
Chapter 3	**MPC1.** Structured requirements metrics suite	– Metric validation framework [Kitchenham, 1995] – Comparative analysis with literature	– Action research [Susman and Evered, 1978; Santos and Travassos, 2011] – Expert opinion [Helmer, 1967]
Chapter 4	**MPC2.** Approach for deriving, analyzing, and oraganizing requirements metrics and related information	N/A	– Example based on real-life project data [Shaw, 2003]
Chapter 5	**MPC3.** Requirements-centric health assessment framework	– Framework built according to theoretical measurement concepts [IEEE, 2017; Staron and Meding, 2018]	– Application on data from 3 real systems projects [Shaw, 2003] – Expert opinion [Helmer, 1967]
Chapter 6	**MPC4.** Meta-model for requirements-driven information for internal stakeholders	– Evaluation by expert opinion that the meta-model adheres to scientific principles of model building [Helmer, 1967; Berenbach et al., 2009]	– Live study at workshop – Expert opinion [Helmer, 1967]
Chapter 7	**MPC5.** R-Pulse: Requirements dashboard prototype	N/A	– Action research [Susman and Evered, 1978; Santos and Travassos, 2011] – Expert opinion [Helmer, 1967]

8.3 Evaluating the Requirements Measurement Program

"For a measurement program to effectively support an organization's goals, it should be scalable, automated, standardized and flexible – i.e. robust" [Staron and Meding, 2016]. In this subsection, we use Staron and Meding's [Staron and Meding, 2016, 2018] measurement program assessment method to assess the robustness of our measurement program. Their method consists of checking the measurement program against criteria in seven categories

[Staron and Meding, 2016, 2018] as listed in Table 8.2.

Table 8.2: Categories for assessment of measurement programs.

Category	**What to Assess**
Metrics organization	Assess how metrics collection, analysis and visualization are done, and by whom.
Metrics infrastructure	Assess how the measurement program is realized technology-wise.
Metrics used	Assess which measures are used in the company or organization.
Decision support	Assess how measures are used in decision processes in the company or organization.
Organizational metric maturity	Assess how the organization, as a whole, works with measures.
Collaboration with academia	Assess the status of research-oriented activities at the organization.
External collaboration	Assess the state of collaboration with other companies.

Before we evaluate the program, we need to note that Staron and Meding's definition of a measurement program encompasses "socio-technical systems where the technology interacts with stakeholders in order to support certain goals" [Staron and Meding, 2016]. Thus, their measurement program evaluation approach assumes the existence of a mature, organization-wide measurement program, which usually requires long periods of time and organizational support to implement and maintain. However, our work is restricted to the requirements engineering process that did not have any form of measurement process or program in place. Thus, our work *lays the foundation* for a mature measurement program.

Metrics organization. Refers to "the metrics team and covers all aspects, e.g. a formal metrics team, a virtual metrics team, or employees working with measuring, independently from each other, in the same company or organization" [Staron and Meding, 2018]. Because the collaborating organization did not implement any form of requirements measurement before the commencement of this work, the metrics team is comprised, thus far, of the author of this work. Thus, the assessment criteria assumes the author of this work as the *metrics team* along with the internal stakeholder who oversaw and supervised the measurement program. Table 8.3 lists the metrics organization/team related criteria and our assessment.

Table 8.3: Assessing the metrics team.

Question	Yes/No/I don't Know
Is there a metrics organization (e.g., team)?	Yes
Does the organization have sufficient resources?	Yes
Is there a metrics organization that maintain existing measures?	Yes
Do measures support the organization with competence?	I don't know
Does the metrics organization give presentations, seminars, and/or courses?	Yes
Does the metrics organization have good knowledge of the company's or organization's products?	Yes
Is it well defined and transparent, how the metrics organization prioritizes its assignments?	No
Can the metrics organization handle emergencies?	I don't know
Does a strategy plan exist?	Yes, using the meta-model (MPC4)
Does a metrics champion exist?	Yes
Does a measurement sponsor exist?	I don't know
Does a measurement analyst exist?	Yes
Does a measurement designer exist?	Yes
Does a measurement librarian exist?	Yes
Does a metrics team leader exist?	Yes
Is there a document describing how the metrics organization works?	Yes, using the meta-model (MPC4)
Is there a contingency plan?	No
Does a statement of compliance towards ISO/IEC/IEE 15939 [IEEE, 2017] exist?	Yes
Does a statement of compliance towards ISO/IEC 12207 exist? [ISO et al., 2015]	No
Does a statement of compliance towards IEE Std 1061 [IEEE, 1992] exist?	Yes
Does a statement of compliance towards ISO/IEC 250xx family exist?	No

Metrics Infrastructure. "The measurement infrastructure forms the technological aspect of the measurement program providing the technical means for realizing the measurement program...and includes the databases for collecting measurement results and execution and dissemination environment." [Staron and Meding, 2016]. Table 8.4 lists the metrics infrastructure related criteria and our assessment of our measurement program in relation to that.

Table 8.4: Assessing the measurement infrastructure.

Question	Yes/No/I don't Know
Does a structure exist that contains all/the most important measures?	Yes, using the structured matrics suite (MPC1) and the dashboard (MPC5)
Is the infrastructure secure?	Yes
Is the infrastructure built up, so that is supports automation?	Yes, through R-Pulse (MPC5)
Do all measurement systems include information quality?	No
Does the infrastructure support/enable dissemination of information products?	Yes, through R-Pulse (MPC5)
Do naming of rules for folders and files exist?	Yes, using the structured metrics suite (MPC1)

Metrics Used. Table 8.5 lists the criteria related to the metrics used in the organization. While we cannot claim that the metrics defined in Chapter 3 (MPC1) address *all* the internal stakeholder information needs, we can say that, because the metrics were derived based on the internal stakeholders' information needs (through action research and using GQM—see Chapter 3 for details), they address a significant portion of their requirements related information needs.

Table 8.5: Assessing the measures used.

Question	**Yes/No/I don't Know**
Are all deployed measures used?	Yes
Are all measures updated with specified frequency?	Yes, through R-Pulse (MPC5)
Can all deployed measures be easily accessed?	Yes, through R-Pulse (MPC5)
Can all information needs be realized?	I don't know

Decision Support. Table 8.6 lists the criteria related to the extent that the measures and indicators support decisions within the organization.

Table 8.6: Assessing the decision support.

Question	**Yes/No/I don't Know**
It is clear/transparent who is interested in the metrics data?	Yes
Are meanings/interpretations of measures defined	Yes
Are measures used for analyses of problems/root causes?	Yes, using the metrics suite (MPC1) and health assessment framework (MPC3)
Are measures and indicators used to formulate decisions?	Yes, using the health assessment framework (MPC3)
Are measures and indicators used to monitor implementation of decisions?	No
Is it clear which measures and indicators, are used for technical and managerial areas respectively?	To an extent using the meta-model (MPC4)

Measurement Maturity. Table 8.7 lists the criteria related to the maturity of the measurement process in the organization. As noted earlier, the requirements measurement program presented in this thesis lays the foundations for a mature measurement program. Despite the nascency of the measurement program, we can see from Table 8.7 that it exhibits a level of maturity through the use of documents and repeatable algorithms, automatic data collection, decision criteria for visualizations, and the measures availability in standard tools.

Table 8.7: Assessing the organization's measurement maturity.

Question	**Yes/No/I don't Know**
Is there a prioritized list of defects per product?	N/A
Is there a list over the most complex software modules?	N/A
Are measures and indicators collected/calculated using documents and repeatable algorithms?	Yes, using the metrics suite (MPC1) and the health assessment framework (MPC3)
Are measures and indicators collected/calculated manually?	Initially
Are measures and indicators collected/calculated automatically?	Yes, using R-Pulse (MPC5)
Are measures and indicators visualized with decision criteria?	Yes, using the RAG system and thresholds in the health assessment framework (MPC3)
Are measures and indicators accompanied with information quality/reliability evaluation?	No
Are measures and indicators available in standard tools (e.g. Eclipse, MS Excel), used and understood in the organization?	Yes, excel and the dashboard (MPC5)

Collaborating with Academia. Table 8.8 lists the criteria related to assessing the collaboration with academia. The organization's collaboration with academia for measurement began with this project. Thus, this work has the potential to promulgate further academic collaboration with the organization in the future.

Table 8.8: Assessing collaboration with academia.

Question	**Yes/No/I don't Know**
Does the metrics organization have collaboration with academia?	Yes
Does the metrics organization execute measurement research projects?	Yes
Does the metrics organization publish papers?	Yes
Does the metrics organization have students on site?	The author of this work and others
Does the metrics organization supervise bachelor/master theses?	No

External Collaboration. Given the nascency of the requirements measurement program in the organization, it has yet to collaborate with other companies for measurement purposes.

The above assessment of the measurement program shows that the requirements measurement program components presented in this thesis contribute to the implementation of a robust measurement program in practice; the tables show how the various measurement program components satisfy a large portion of the listed robustness criteria. However, this work is but a a first step towards establishing a mature requirements measurement program for the collaborating organization and other organizations.

References

[Berenbach et al., 2009] Berenbach, B., Paulish, D. J., Kazmeier, J., and Rudorfer, A. (2009). *Software and systems requirements engineering in practice*. McGraw Hill.

[Helmer, 1967] Helmer, O. (1967). Systematic use of expert opinions. Technical report, The RAND Corporation, Santa Monica, California.

[IEEE, 1992] IEEE (1992). IEEE 1061 standard for a software quality metrics methodology. Technical report, IEEE.

[IEEE, 2017] IEEE (2017). ISO/IEC/IEEE 15939:2017(E) - Systems and software engineering - Measurement Process. Technical report, ISO/IEC/IEEE.

[ISO et al., 2015] ISO, IEC, and IEEE (2015). ISO/IEC/IEEE 15288 - Systems and Software Engineering - System life cycle processes. Technical report, ISO, IEC, IEEE.

[Kitchenham, 1995] Kitchenham, B. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944.

[Santos and Travassos, 2011] Santos, P. S. M. d. and Travassos, G. H. (2011). Action research can swing the balance in experimental software engineering. *Advances in Computers*, 83:205–276.

[Shaw, 2003] Shaw, M. (2003). Writing good software engineering research papers. In *25th International Conference on Software Engineering (ICSE 2003)*, pages 726–736, Portland, Oregon. IEEE.

[Staron and Meding, 2016] Staron, M. and Meding, W. (2016). MeSRAM - A method for assessing robustness of measurement programs in large software development organizations and its industrial evaluation. *Journal of Systems and Software*, 113:76–100.

[Staron and Meding, 2018] Staron, M. and Meding, W. (2018). *Software development measurement programs: Development, management and evolution*. Springer.

[Susman and Evered, 1978] Susman, G. and Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative Science Quarterly*, 23(4):582–603.

Chapter 9

Conclusions and Future Work

To conclude this thesis, we revisit the problem and our research goal from Chapter 1, summarize our contributions in light of the research goal, and reflect on our contributions and draw conclusions from them in Section 9.1. Finally, we discuss future work in Section 9.2.

9.1 Summary and Conclusions

Measurement in systems and software engineering began garnering serious attention in the software engineering community since the 1970s [Staron and Meding, 2018]. Since then, it has received significant attention due to the many benefits accrued from measuring software and systems processes, products, and projects. The effort dedicated towards measurement in software engineering takes the form of software metrics [Chidamber and Kemerer, 1994], metric validation [Kitchenham, 1995; Briand et al., 1995], metric derivation approaches [Basili et al., 1994], and measurement tool support [Johnson, 2007; Sharma and Kaulgud, 2012; López et al., 2018], to name a few.

Similarly, requirements measurement has shown to be beneficial in tracking development progress, identifying gaps in the downstream deliverables related to requirements [Kratschmer, 2013], managing risks that may be introduced due to late changes to requirements [Costello and Liu, 1995; Kratschmer, 2013], reducing requirements errors and defects [Davis et al., 1993; Costello and Liu, 1995], and in aiding in project management and decision making [IEEE, 2017].

However, the bulk of the effort has been for code-centric measurement. Specifically, only 5% of the literature on metrics addresses the requirements phase of a system or software project. While the benefits of requirements measurement have been recognized by researchers [Costello and Liu, 1995] and practitioners [Kolde, 2004; Wiegers, 2006; Kratschmer, 2013], there is a lack of effort regarding the metrics, methods, and tools needed to establish requirements measurement programs. Our experience with industry echoes this deficit; despite the need within the company for requirements measurement, practice and literature do not provide the tools necessary to implement requirements measurement.

Using the gap in the literature and our experience with industry as our motivation, our aim in this thesis was to create a requirements measurement program that would begin filling the gap in the literature and addressing the need in practice. Thus, in collaboration with an organization in the rail automation domain, we created the following solutions for requirements measurement, each reported in a standalone chapter:

1. A structured requirements metric suite that consists of: (i) a set of five measurable requirements attributes, (ii) four novel metric levels, (iii) 90 requirements metrics measuring each of the five attributes at the four metric levels, and (iv) 9 requirements meta-data items to apply the metrics (Chapter 3).

   The metrics suite provides practitioners with the foundational measurement elements for applying requirements metrics and reporting measures in the software and systems development processes so as to provide internal stakeholders with the necessary requirements-driven information.

   Within the realm of research, the contributions of this chapter bring an unprecedented level of scientific rigor to requirements metrics that is currently lacking in the literature, which is achieved by: providing formal definitions of the metrics and validating them theoretically and empirically, shedding light on the scientific properties of measurable requirements attributes, introducing novel requirements metrics levels, and pushing towards establishing a consensus on the required meta-data items needed to facilitate the requirements measurement process.

   The work in this chapter led to several conclusions. First, requirements and requirements meta-data provide fertile ground for defining and applying useful requirements metrics that can be incorporated into the requirements management and development processes. Second, requirements metrics do, in fact, facilitate the requirements management process and provide internal stakeholders with requirements-driven information that aided them in carrying out their development related tasks. Such tasks included ensuring requirements-design coverage by architects, release planning by systems and products managers, safety requirements management by safety managers, and communication to upper management and clients. Finally, different measures for the various attribute-level combinations are used differently by internal stakeholders.

2. An approach to derive and organize requirements metrics and related information (Chapter 4). The approach consists of 7 steps and combines the use of GQM [Basili et al., 1994] and the identification and analysis of four main RE measurement elements: attributes, levels, metrics, and meta-data items. The contributions of this chapter provide a means to achieve the results in Chapter 3.

   We applied the approach to real-life projects and showed how the approach led to a more comprehensive set of requirements metrics, improved organization and reporting of metrics, and improved consistency, completeness of requirements meta-data across projects, and reduction of requirements measurement time.

   The approach and its application demonstrate that, despite the significant attention given to software engineering measurement, there exists measurement challenges specific to requirements, particularly in a large systems engineering context. Thus, the approach is but a step towards addressing those challenges through further research and investigation of requirements measurement approaches.

3. A requirements-centric project health assessment framework that measures and analyzes critical requirements attributes, in conjunction with project data (e.g., deadlines and resources), and identifies their health indicators, which are visualized using a RED-AMBER-GREEN (RAG) indicators system (Chapter 5). The resultant health indicators provide a high-level view of project health from a requirements perspective with the ability to drill down to specific, raw measures. The proposed framework in this chapter builds upon the results (i.e., requirements attributes and metrics) identified in Chapter 3.

   The framework aids manual and personal-based assessments with a quantitative and systematic assessment of project health that can aid in identifying critical requirements-related problems that may negatively impact the project's likelihood of success.

   The contributions of this chapter show that there is a need for methods and tools that integrate requirements into project health assessments. Our framework demonstrates that such an endeavour is feasible and worthwhile as it supports manual and personal assessments with quantitative, requirements-centric information, which, in turn, aids in managing and monitoring large systems projects more effectively.

4. A meta-model that seeks to make sense of web of interactions among measurement entities in a project or organization in order to facilitate the management of the requirements measurement process (Chapter 6). Specifically, the meta-model maps out the entities and relationships involved in providing requirements-driven information

to internal stakeholder within the context of a systems project. The meta-model consists of three abstraction levels in which the first (highest) abstraction level consists five entities and nine relationships. The entities and relationships are further decomposed in the lower abstraction levels.

The benefits of using the meta-model include i) control and management of processes and resources involved in providing requirement-driven information to internal stakeholders, ii) communication among internal stakeholders, and iii) aligning internal stakeholder concerns with requirements-driven information that can be generated within the project.

For research, the meta-model enhances our understanding of stakeholders' information needs through identifying the different types of stakeholders in relation to requirements and requirements specifications.

5. A web-based requirements dashboard (R-Pulse) prototype that automates the concepts from the previous chapters 7. R-Pulse uses exports of the requirements documents and other software artifacts (e.g., design, test, and defects) to calculate the requirements metrics and health indicators. Internal stakeholders are able to view the project's health indicators and drill down to the requirements measures. The dashboard provides various visualization and navigation options so as to address each internal stakeholder's need with regard to the requirements-driven information.

The contributions were validated using various validation techniques such as action research [Santos and Travassos, 2011], focus groups , expert opinion [Yousuf, 2007; Helmer, 1967], which we summarize in Chapter 8. Moreover, we use established measurement program assessment criteria [Staron and Meding, 2018] to evaluate the robustness of the overall requirements measurement program. Our assessment shows that our measurement program satisfies the majority of the robustness criteria for measurement programs.

It remains the case that this research was conducted within a specific context, that is, the context of a large systems engineering project in the rail-automation domain. Such a domain is characterized by mature requirements engineering software development processes, organization and management-level support for measurement, and sophisticated requirements infrastructure, all of which contributed to making a conducive environment for a requirements measurement program. The downside, however, is that generalizing the findings and solutions of this thesis must be done with caution. On the other hand, this work lays a solid foundation for further work and research on requirements measurement, which we discuss in the following section.

9.2 Future Work

Despite the progress made in this thesis towards developing the field of requirements measurement, there remains ideas and areas worth exploring in the future. This section discusses some potential future paths.

As discussed earlier, the solutions presented in this thesis are rooted in industrial practice and were built to address real-world problems in a specific domain with waterfall-like software development processes. Thus, an important area for future work consists of investigating the applicability of our solutions in other development environments (e.g., agile processes) and domains (e.g., health, banking and web services) to improve its generalizability. Some questions that are worth asking include: Are the same set of requirements metrics useful in other domains? What are the internal stakeholders' information needs with regard to requirements in an agile process? Does the meta-model capture the measurement process in other software development processes?

It is expected, then, that the proposed solutions would require customization for different domains and development processes. Thus, studying the customization options of the proposed solutions in order for them to be useful and effective in different domains and development life cycles is another important line of future work.

Another area of future research is integrating the concepts in this thesis (i.e., metrics, health indicators, attribute-level combinations, internal stakeholder information needs, and requirements dashboard) into current requirements tools such as DOORS, Modern Requirements, Jama Software, and ReqSuite, to name a few. This would allow for a seamless requirements measurement and management process and would reduce the overhead associated with learning and adopting requirements measurement concepts and activities.

Finally, the requirements metrics, approach, model, and tool in this thesis are geared towards addressing the needs of internal stakeholders such as requirement engineers, architects, R&D managers, developers, and so forth. However, external stakeholders and users (e.g., contractors, regulatory bodies, and customers, etc.) have a vested interest in the system and software requirements as well. Exploring the ways requirements measurement can be utilized to address the needs of external stakeholders is another future endeavor worth undertaking.

References

[Basili et al., 1994] Basili, V., Caldiera, G., and Rombach, H. (1994). Goal question metric approach. *Encyclopedia of Software Engineering*, 1:98–102.

[Briand et al., 1995] Briand, L., El Emam, K., and Morasca, S. (1995). Theoretical and empirical validation of software product measures. Technical report, International Software Engineering Network.

[Chidamber and Kemerer, 1994] Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

[Costello and Liu, 1995] Costello, R. J. and Liu, D.-B. (1995). Metrics for requirements engineering. *Journal of Systems Software*, 29:39–63.

[Davis et al., 1993] Davis, A., Overmeyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., and Theofanos, M. (1993). Identifying and measuring quality in a software requirements specification. In *Proceedings of the 1st International Software Metrics Symposium*, pages 141–152, Baltimore, MD, USA. IEEE.

[Helmer, 1967] Helmer, O. (1967). Systematic use of expert opinions. Technical report, The RAND Corporation, Santa Monica, California.

[IEEE, 2017] IEEE (2017). ISO/IEC/IEEE 15939:2017(E) - Systems and software engineering - Measurement Process. Technical report, ISO/IEC/IEEE.

[Johnson, 2007] Johnson, P. M. (2007). Requirement and design tradeoffs in Hackystat: An inprocess software engineering measurement and analysis system. In *1st International Symposium on Empirical Software Engineering and Measurement*, pages 81–90, Madrid, Spain. IEEE.

[Kitchenham, 1995] Kitchenham, B. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944.

[Kolde, 2004] Kolde, B. C. (2004). Basic metrics for requirements management. Technical Report March, Borland.

[Kratschmer, 2013] Kratschmer, T. (2013). Requirements and metrics. *Requirements Management Blog - IBM*. www.ibm.com/developerworks/community/blogs/requirementsmanagement/entry/requirements-metrics?lang=en.

[López et al., 2018] López, L., Martínez-Fernández, S., Gómez, C., Choraś, M., Kozik, R., Guzmán, L., Vollmer, A. M., Franch, X., and Jedlitschka, A. (2018). Q-rapids tool prototype: Supporting decision-makers in managing quality in rapid software development. In *Lecture Notes in Business Information Processing*, volume 317, pages 200–208.

[Santos and Travassos, 2011] Santos, P. S. M. d. and Travassos, G. H. (2011). Action research can swing the balance in experimental software engineering. *Advances in Computers*, 83:205–276.

[Sharma and Kaulgud, 2012] Sharma, V. S. and Kaulgud, V. (2012). PIVoT: Project insights and visualization toolkit. In Zurich, S., editor, *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM'12)*, pages 63–69. IEEE.

[Staron and Meding, 2018] Staron, M. and Meding, W. (2018). *Software development measurement programs: Development, management and evolution.* Springer.

[Wiegers, 2006] Wiegers, K. E. (2006). *More about software requirements: Thorny issues and practical advice.* Microsoft Press, Washington.

[Yousuf, 2007] Yousuf, M. I. (2007). Using experts' opinions through Delphi Technique - Practical assessment, research and evaluation. *Practical Assessment, Research and Evaluation*, 12(4).

Appendix A

Exploratory Interactive Study Details

The interactive study took place at the International Empirical Requirements Engineering (EmpiRE'18) Workshop co-located with the International Requirements Engineering Conference. A version of this appendix was published in [Noorwali and Madhavji, 2018].

Two major activities were carried out in the study: (i) gathering relevant data for model enhancement, and (ii) preliminarily analysis of the data while involving the participants. Below I describe the research questions posed to the participants and the study design.

A.1 Research Questions

With reference to the preliminary model,

- RQ1. Which additional elements do you think would enhance the model?
- RQ2. Which elements do you think can be removed from the model?
- RQ3. Which elements do you think can be modified from the model?
- RQ4. Which additional relationships among the elements do you think would enhance the model?
- RQ5. Which relationships among the elements do you think can be removed from the model?
- RQ6. Are there any complimentary considerations that have not been treated by the existing model (e.g., assumptions, rationales, etc.)?
- RQ7. Which relationships among the elements do you think can be modified from the model (e.g., direction, cardinality, etc.)?

A.2 Design of the Exploratory Interactive Study

The main idea of the study is to pose the research questions to the participants —displayed in a large and accessible format—and allow them to provide answers in an interactive and simultaneous manner by posting their answers onto clusters of common areas on a *wall-of-ideas* (e.g., one cluster per research question). Details of the study procedure are as follows:

Introduction: 20-25 minutes:

- The researcher-moderator presented the current research, results, concepts, context, and the preliminary domain model to the participants.
- The researcher-moderator gave a briefing on the domain model construction process.
- The researcher-moderator presented and explained the research questions (RQs) to the participants and placed the RQs on a board with space for participant contributions.
- Post-it notes and pens were made available for participants to make their contributions.

Data gathering: 25-30 minutes

- The researcher-moderator then opened the floor to the participants to provide possible answers for the RQs. The answers to each RQ were posted in their designated areas on the *wall-of-ideas*.
- There was also a fourth area for the participants to provide comments and feedback outside the scope of the seven RQs.
- During data gathering, the researcher-moderator was standby for any needed assistance.

Data analysis: 25-35 minutes

- Data analysis consisted mainly of qualitative data analysis. The researcher-moderator lead the analysis of the participants' answers (post-it notes) in a plenary manner and discussed selected responses by:
  - Selecting heavily loaded clusters of responses and discuss rationale behind these post-it notes (source participants will be invited to make comments).

- Selecting empty or lightly loaded clusters and discuss triggers for new elements and relationships.
- Discussing generalizability of the overall model.
- Performing a comparative analysis of the 'before' and 'after' domain models.

References

[Noorwali and Madhavji, 2018] Noorwali, I. and Madhavji, N. H. (2018). A domain model for requirements-driven insight for internal stakeholders: A proposal for an exploratory interactive study. In *7th IEEE International Workshop on Empirical Requirements Engineering (EmpiRE 2018)*, pages 32–36. IEEE.

Appendix B

Questionnaire for the Evaluation of the Meta-Model for Requirements-Driven Information for Internal Stakeholders

B.1 Introduction and Instructions

- This document depicts a meta-model constructed based on the results of an action research study conducted in a large systems project in the rail automation domain.
- The purpose of this document is to validate the meta-model regarding its usefulness with perspectives from the field of practice.
- Below, please find a questionnaire which we kindly request your responses to. The questionnaire is composed of six questions (three background questions and three meta-model specific questions). We anticipate that it would take no more than 10 minutes to answer all the questions. All opinions are valid and will be gratefully respected.
- This document is composed of two sections: Section 1 presents a brief description of the meta-model, and Section 2 presents the validation questions.

B.2 Section 1: Brief Description of the Meta-Model

Providing requirements-driven information (e.g., requirements volatility measures, requirements-design coverage information, and requirements growth rates, etc.) falls within the realm of the requirements management process. The requirements engineer must derive and present the appropriate requirements information to the right internal stakeholders in the project. This process is made complex due to project-related factors such as numerous types of internal stakeholders, varying stakeholder concerns with regard to requirements, large project sizes, a plethora of software artifacts, and many affected processes. However, currently, there is little guidance in practice as to how these factors come into play together in providing the described information to the internal stakeholders. Based on analyzed data from an action research study we conducted in a large systems project in the rail-automation domain, we propose a meta-model that consists of the main entities and relationships involved in providing requirements-driven information to internal stakeholders within the context of a large systems project. The meta-model consists of five main entities and nine relationships that are further decomposed into three abstraction levels. The meta-model is anticipated to facilitate: (i) control and management of process and resources for providing requirement-driven information to stakeholders; (ii) communication among internal stakeholders; and (iii) reuse of the meta-model across different projects. The proposed meta-model is depicted in Figure B.1.

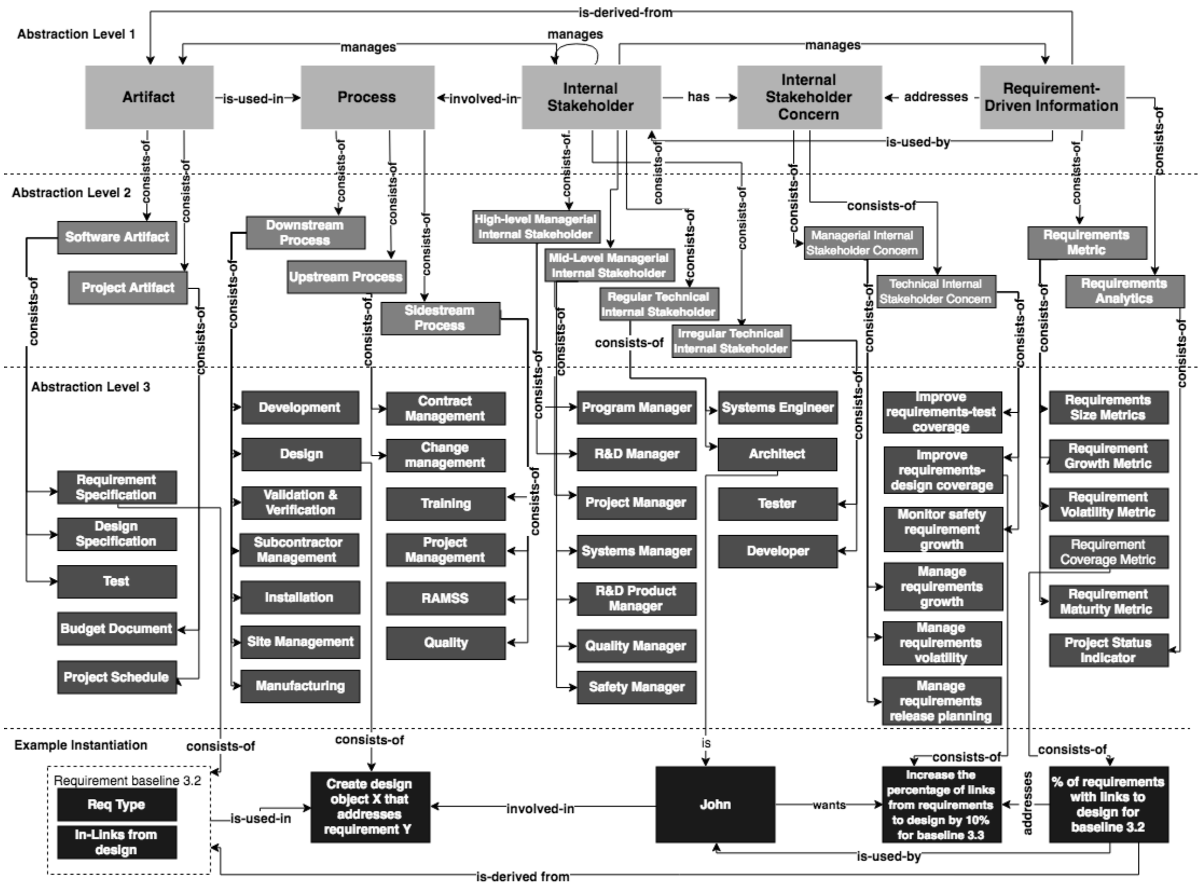

Abstraction Level 1
is-derived-from
manages
manages
manages
Artifact
is-used-in
Process
involved-in
Internal Stakeholder
has
Internal Stakeholder Concern
addresses
Requirement-Driven Information
is-used-by
consists-of
Abstraction Level 2
Software Artifact
Project Artifact
Downstream Process
Upstream Process
Sidestream Process
High-level Managerial Internal Stakeholder
Mid-Level Managerial Internal Stakeholder
Regular Technical Internal Stakeholder
Irregular Technical Internal Stakeholder
Managerial Internal Stakeholder Concern
Technical Internal Stakeholder Concern
Requirements Metric
Requirements Analytics
Abstraction Level 3
Development
Design
Validation & Verification
Subcontractor Management
Installation
Site Management
Manufacturing
Contract Management
Change management
Training
Project Management
RAMSS
Quality
Program Manager
R&D Manager
Project Manager
Systems Manager
R&D Product Manager
Quality Manager
Safety Manager
Systems Engineer
Architect
Tester
Developer
Requirement Specification
Design Specification
Test
Budget Document
Project Schedule
Improve requirements-test coverage
Improve requirements-design coverage
Monitor safety requirement growth
Manage requirements growth
Manage requirements volatility
Manage requirements release planning
Requirements Size Metrics
Requirement Growth Metric
Requirement Volatility Metric
Requirement Coverage Metric
Requirement Maturity Metric
Project Status Indicator
Example Instantiation
consists-of
is
Requirement baseline 3.2
Req Type
In-Links from design
is-used-in
Create design object X that addresses requirement Y
involved-in
John
wants
Increase the percentage of links from requirements to design by 10% for baseline 3.3
addresses
% of requirements with links to design for baseline 3.2
is-used-by
is-derived from

Figure B.1: Meta-model for requirements-driven information for internal stakeholders.

With reference to Figure B.1, we then discuss the entities and relationships presented in the model. Table B.1 provides a short description of the entities depicted in the proposed model.

Table B.1: Descriptions of meta-model entities at abstraction level 2.

Entity	**Description**
Requirements Metric	A measurement derived from requirements to provide a quantitative assessment of certain requirements attributes
Requirement Analytics	Analytics on requirements data in conjunction with other software artifacts (e.g., design, code, budget and schedule documents, etc.) that aims to gain insight about the state of the project from a requirements perspective
High-level Managerial IS	Managerial stakeholders who manage at the project level or higher (i.e., program or regional levels) such as the program or regional R&D manager, etc.
Mid-Level Managerial IS	Managerial stakeholders who manage at the project level or lower (i.e., product level) such as test manager, product quality manager, etc.
Regular Technical IS	Technical ISs who use requirement-driven information regularly such as architects and requirements engineers
Irregular Technical IS	Technical internal stakeholders who use requirement-driven infor-mation less frequently such as developers and testers
Managerial IS Concern	Managerial issues that ISs care about in relation to the requirements such as estimating time and effort for a software release
Technical IS Concern	Technical issues that ISs care about in relation to the requirements such as increasing requirement-design coverage
Downstream Process	Activities involved in system development and initiated after the requirements engineering process such as development, design, testing, etc.
Upstream Process	Activities that are involved in system development and are initiated before the RE process such as contract/client management
Side-stream Process	Activities involved in system development and initiated and executed alongside the RE process such as quality and project management, etc.

There are three abstraction levels depicted in this model:

- **Level 1:** Entities and relationships at level 1 are abstract and generalizable enough to be applicable to any context regardless of domain, software development process, or organizational structure.
- **Level 2:** Entities at abstraction Level 2 are also intended to be generalizable to different contexts. However, its applicability may differ from one context to another. The entities at abstraction level 2 are further decomposed into entities at abstraction Level 3.
- **Level 3:** The project specific level in which the entities are tailored to represent the environment of a given project in different domains and development processes. For example, requirement metrics in our study consisted of size, growth, volatility, coverage, and maturity metrics. Another project's requirements metrics may include only volatility metrics. The same applies to other entities. Because entities at level 3 are specific to our project, we did not include a detailed description of them. However, they illustrate how the meta-model can be applied within a large systems project.

B.3 Section 2: Validation Questions

B.3.1 Background Questions

1. What is your job title?

2. How many years of experience do you have in industry? And how many years of your experience in industry are in requirements engineering?

3. How many years of experience do you have with research? And how many years of experience in research are in requirements engineering?

B.3.2 Meta-Model Feedback Questions

4. To what extent do you agree that the meta-model is useful for practice?

1. Strongly disagree
2. Disagree

3. Disagree
4. Neither agree or disagree
5. Agree
6. Strongly agree

Justify your opinion:

5. If you were to use this meta-model in your project, how will it be useful?

6. Any general comments (please kindly provide any other recommendations below for improving of the meta-model)?

Curriculum Vitae

Name:	Ibtehal Noorwali
Post-Secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada Ph.D., Computer Science (2013–2020)
	Huron College at The University of Western Ontario London, Ontario, Canada M.A., Theology (2014–2019)
	The University of Western Ontario London, Ontario, Canada M.Sc., Computer Science (2011–2013)
	Umm Al-Qura University Makkah, Saudi Arabia B.Sc., Computer Science (2005–2009)
Honours and Awards:	Dr. Sadika Merchant Kidwai and Mr. Arshed Ali Kidwai Scholarship 2019
	Government of Saudi Arabia - Ministry of Education Scholarship for Higher Education 2011–2019
Related Work Experience:	Teaching Assistant The University of Western Ontario–London, Ontario, Canada 2019–2020
	Systems Engineering Intern and Collaborative Researcher Siemens Mobility Inc.–New York, NY, USA 2017
	University Instructor Umm Al-Qura University–Makkah, Saudi Arabia 2009–2011

Publications:

1. **Noorwali, I.**, Madhavji, N. H., and Ferrari, R. (2020, May). An approach for deriving, analyzing, and organizing requirements metrics and related information in systems projects. In *15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'20).* Prague, Czech Republic (to appear).

2. Arruda, D., Madhavji, N. H., and **Noorwali, I.** (2019, July). A validation study of a requirements Engineering artefact model for big data software development projects. In *14th International Conference on Software Technologies (ICSOFT'19)*, pages 106-116, Prague, Czech Republic. SciTePRess.

3. **Noorwali, I.**, Madhavji, N. H., Arruda, D., and Ferrari, R. (2019, March). Towards a meta-model for requirements-driven information for internal stakeholders. In Knauss, E. and Goedicke, M., editors, *25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'19)*, volume 1, pages 262–278, Essen, Germany. Springer Nature.

4. **Noorwali, I.** and Madhavji, N. H. (2018, August). A domain model for requirements-driven insight for internal stakeholders: A proposal for an exploratory interactive study. In *7th IEEE International Workshop on Empirical Requirements Engineering (EmpiRE'18)*, pages 32–36, Banff, AB, Canada. IEEE.

5. **Noorwali, I.** (2018, March). Stakeholder concern-driven requirements analytics. *ACM SIGSOFT Software Engineering Notes*, 43(1), 1-6.

6. **Noorwali, I.**, Arruda, D., and Madhavji, N. H. (2016, May). Understanding quality requirements in the context of big data systems. In *2nd International Workshop on BIG Data Software Engineering (BIGDSE'16)*, pages 76-79, Austin, Texas, USA. ACM.

7. Nekvi, M. R. I., **Noorwali, I.**, and Madhavji, N. H. (2016, March). Deriving metrics for estimating the effort needed in requirements compliance work. In *22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'16)*, pages 135-141, Gothenburg, Sweden. Springer.

8. Ferrari, R., **Noorwali, I.,** and Madhavji, N. H. (2015, August). Towards a theory on the interaction between requirements engineering and systems architecting. In *5th IEEE International Workshop on Empirical Requirements Engineering (EmpiRE'15)*, pages 17-20, Ottawa, Canada. IEEE.

9. **Noorwali, I.** and Madhavji, N. H. (2013, April). Maps of lessons learnt in requirements engineering: a research preview. In *19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'13)*, pages 119-124, Essen, Germany. Springer, Berlin, Heidelberg.