A VALUE AND DEBT AWARE FRAMEWORK FOR EVALUATING COMPLIANCE IN SOFTWARE SYSTEMS

by

BENDRA ELOHO OJAMERUAYE

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
May 2016

Abstract

Today's software systems need to be aligned with relevant laws and other prevailing regulations to control compliance. Compliance refers to the ability of a system to satisfy its functional and quality goals to levels that are acceptable to predefined standards, guidelines, principles, legislation or other norms within the application domain. In some domains, compliance is a primary driver for eliciting and verifying requirements. Ensuring systems are legally compliant is expensive and challenging for software engineers. Compliance requirements can be viewed as a concern, which cuts across different functional and non-functional requirements with significant impact on the architecture. Addressing compliance requirements at an early stage of software development is vital for successful development as it saves time, cost, resources and the effort of repairing software defects. Compliance requirements are usually an afterthought and seen as not creating value; hence it is not properly built into systems. We argue that the management of compliance and compliance requirements is ultimately an investment activity that requires value-driven decision-making. The work presented in this thesis revolves around improving decision support for compliance by making them value, risk and risk aware. It also concretely defines and breaks down the concept of value as it pertains to compliance and the development of software systems. This thesis presents a value-driven framework for the systematic treatment of compliance requirements in software systems. Specifically, this thesis presents an economics-driven approach, which leverages on goal-oriented requirements engineering with portfolio-based thinking and technical debt analysis to enhance compliance related decisions at design-time. The approach is value driven and systematic; it leverages on influential work of portfolio thinking and technical debt to make the link between compliance requirements, risks, value and debt explicit to software engineers. The approach is evaluated with two case studies to illustrate its applicability and effectiveness.

DEDICATION

In loving memory of Dr. Ben Ojameruaye (1958 - 2016)

Father. Friend. Hero

To memory of my dear father, Dr Ben Ojameruaye, for instilling the virtues of strength and perseverance in the face of difficulties, for giving me a good foundation. For being a role model of resilience, strength and character. You taught me about hard work, self-respect and independence. You worked hard to educate us, without which this degree would have been impossible.

"It is God that gave me strength, and made my way perfect." Psalms 18:32

ACKNOWLEDGEMENTS

With a debt of gratitude, which cannot be adequately expressed in words, I thank my supervisor Dr Rami Bahsoon. Thanks for being a friend, an excellent listener and for the guidance, encouragement, patience, commitment to quality and endless support. This dissertation could not have been written without your supervision, and support. I have been very fortunate to work with you. I would also like to thank my thesis group members, Prof. Reddy Uday and Dr. Flavio Gracia for their insightful comments.

Life in Computer Science has been a great pleasure due to the friends that I made during the course of PhD. I really enjoyed discussions and gossips with Shen, Funmi, Momoduo, Guanbo, Carlos and Esra. Lots of thanks to David for proofreading drafts of this thesis and being a reliable friend.

I sincerely thank Dr Kashif Rajpoot and Dr Raian Ali for the time and effort in examining this thesis. The discussions during the viva and feedback has helped to significantly improve the thesis.

Endless thanks to my parents for everything and the strength to reach for the stars. I am grateful to them for the ”smart genes” they passed on to me. To my mother, thank you for everything. To daddy, I am fortunate to have had an amazing father and the time that we spent together will always be cherished. Ovo, Kevwe, Ighome, Oke (my main support system), Akpevwe, Fejiro, words cannot describe how much I appreciate you all. Special thanks to my husband, Victor, my tailwind. To Zarah and Zoe, my pride and joy, this work should serve as a testament to perseverance. Giving birth to two kids during the PhD wasn’t easy but you both have given me blessings, hope and happiness.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

”Imagination is more important than knowledge.

Knowledge is limited. Imagination encircles the world.”

Albert Einstein

1.1 Motivation

Today’s software systems need to be aligned with relevant laws and other prevailing regulations to control compliance. Regulatory compliance refers to an organization’s responsibility to operate in agreement with established laws, regulations, standards, and specifications [101]. In some domains, compliance is a primary driver for eliciting requirements. For example, compliance has been the primary driver of information security in industry since 2005 [186]. It is difficult to build software systems that comply with regulations [133]; [146] . The need for compliance arises when stakeholders establish that there is a need to operate in agreement with established laws, regulations, standards, and specifications [103] to protect themselves from any vulnerabilities, risk or loss of value involving the consequences of non-compliance. Compliance requirements express this need, describing the risk to be prevented. To ensure software systems are designed to be compliant, software engineering must specify and design systems that satisfy compliance concerns.

Requirement Engineering (RE) is an important branch of Systems Engineering (SE), which focuses on finding approaches for discovering, analysing, specifying, and managing software requirements [168]. The goal of requirement engineering is to make sure that developed product meets the expectations and needs of a customer [169]; [115]. Value-based RE [92] has the same goal – to deliver the right set of functionalities while linking the decisions to value. The objective of addressing compliance requirements with a value perspective is to identify optimal decision choices and explore trade-off decision-making to satisfy compliance while maximising value.

Requirements can be classified into functional and quality requirements. Functional requirements describe what the system will do, while quality requirements describe how well the software will perform [58]. Systems are built mainly to solve real-world problems and the realisation of the functional requirements are constrained quality requirements, which play an important role in design decisions. Non-functional requirements are mainly associated with user satisfaction [180] and business goals [58]; [117], and they are the significant indicators of software success. We view compliance requirements as a concern, which cuts across different functional and non-functional requirements. These requirements do not have simple true or false satisfaction criteria; rather their level of satisfaction can vary with respect to the application domain and system design. Addressing these requirements at an early stage of software development is vital for successful development as it saves time, cost, resources and the effort of repairing software defects [57].

Although compliance requirements are crucial to the system sustainability, they do not have clear link to revenue generation. Therefore, the benefits and returns of compliance investments are difficult to comprehend and visualize. The value is usually questioned and the situation is intensified in projects that must balance very limited resources. Satisfying a compliance requirement can depend on the risk value attached to not complying with that requirement. Furthermore, compliance is difficult to measure as it can cut across many concerns within a system. This makes the measurement for compliance hard to simplify and bound the problem space. Also, compliance involves a dynamic mix of changing regulations, interaction between different stakeholders in the organisation. Another challenge, which faces requirement engineers is ensuring that the specified compliance requirements are neither too idealist nor too weak with respect to business goals as well as finding trade-offs between achieving compliance requirements and the available resources.

Software architecture is the high-level structure and organisation of a software system. An architecture is a set of design decisions that intends to meet all of the functional requirements, while optimising desired quality attributes [173]. Problems in developing software system architectures are often due to an incomplete knowledge of the system, its environment and capacities. It can be also due to poor estimates or misconceptions about the system requirements, system environment and external conditions. The process of architectural design implies a fit between the requirements, system conditions and constraints. Incomplete information and uncertainty may increase the cost of the architecture, introduce risks, alter its value and influence the extent to which it can evolve and sustain over time. Software engineers often neglect to consider that many of the design problem parameters are not completely known or deterministic. As a result, they often fail to recognise that their final design may not have completely met the actual requirements. Compliance requirements should not be managed in isolation of the architecture, because decisions related to compliance propagate to the architecture. The architecture is another level of abstraction for evaluating compliance.

Engineering compliance requirements and designing architectures that satisfy these requirements are intertwined encompassing both technical and value-based (cost, value and risk) decision making. Decisions taken in the requirement space, cut to the architectural space. Compliance requirements should not be managed in isolation of the architecture, because any decision related to compliance can propagate to the architecture. The architecture is another level for evaluating compliance concerns. Linking technical decisions to value-based reasoning facilitates a better understanding for the risks and likely value, when relating requirements to architectures in the presence of uncertainty. Economics-driven software engineering has acknowledged that integrating value-based theories with the technical decision-making can yield measurable improvements in development cost, time and risk management [30].

The process of architecting software to satisfy compliance requirements can be value-aware. This is not easy as compliance needs to be viewed from different dimensions and perspectives. The analysis can be complex, as it requires long term strategic thinking that caters for risks and uncertainty. In terms of compliance, risk can be defined as the potential of loss from non-compliance. This use of this definition is limited in a principle-based environment as uncertainty attached to arrangements that are needed to satisfy compliance requirements. This means that compliance risk is wider than the regulatory violations and breaches. This risk can be analysed from different of perspectives, including an economic and technical perspective.

Uncertainty is inevitable at the early stages of software development when decisions need to be made about the overall organisation of a software system [121]. In general, these decisions aim at maximising the benefits that the software system will bring to its stakeholders, subject to constraints. Uncertainty includes uncertainty about the impact of alternative strategies on compliance goals, the practicality, cost, and risks associated with these alternatives, as well as future changes in compliance requirements. When making decisions, uncertainty is the lack of complete knowledge about the actual consequences of each alternative [121].

With limited resources and uncertainty, adding compliance risks should be value-driven to ensure that the strategy to satisfy these compliance requirements is acceptable. The costs of compliance needs to be proportionate with the level of compliance risk. It is important to manage risk to better allocate resources to achieve an optimum compliant architecture – one aimed at achieving the best compliance outcome for the resources used. In risk management term, this involves systematic identification, assessment and ranking of the risks associated with compliance. Decisions concerning accepting compliance risks need to be based on quantitative and qualitative criteria.

We argue that the management of compliance and compliance requirements is ultimately an investment activity that requires value-driven decision-making and the decision may come with a technical debt. Technical debt can span several dimensions, which are related to selection decision or architecture. For example, technical debt can be related to overdesigning or under-designing for compliance. In addition, technical debt can present in situations where the selection decisions are not fully justified for maximising value [6]. There is need for a value-based approach for managing compliance, to allow organisation adequately analyse legal regulations and achieve compliance while accounting for compliance risk as a form of technical debt.

Consider the following motivating scenario, which illustrate the need for a value-based approach to managing compliance requirements. Agency B is a Federal Agency with a decade old, off-the-shelf customer order tracking and fulfilment system, which is financially unsupportable and no longer meeting the agency's needs. Agency B is looking to a cloud-based software-as-a-service customer management system and e-commerce solution to collect and fulfil sales over an e-commerce solution.

1. Customers will provide personal information as part of the process of registering for the public facing store front.
2. Customers will enter order and payment details into the website
3. The data is processed and the payment is verified by service provider
4. The results of the order are stored on a financial management system

Card payments will be made on the web-site's e-commerce store; hence compliance with the PCI/DSS standard as well as the Data Protection Act are essential requirements.

For cloud-based provision, transparency,location of data centres, availability and loss of control over data security are legitimate security challenges. Cloud providers not willing to take complete responsibility for Information Security. There is need for an approach for managing compliance requirements, to allow organisation adequately analyse legal regulations and achieve compliance using the "best" value. This approach will allow for the use of value-based approaches to understand, communicate and manage issues of benefits, costs, trade-offs, gap analysis, risk management and decision-making while comparing different viewpoints to achieve the required compliance at the best cost. Modelling the required compliance and evaluating different options at an early stage will help in the technical decision-making and it will be a cost-effective approach to create value under uncertainty and to mitigate the risks of unjustified costs due to wrong decisions.

From the above scenario, we can infer that most organisations have some requirements, and these requirements become more important if they have to comply with some standards. Compliance requirements are usually an afterthought and seen as not creating value; hence it is not properly built into systems. The work presented in this thesis revolves around improving decision support by making them value and risk aware. It also concretely defines and breaks down the concept of value as it pertains to compliance and the development of software systems. This thesis presents a value-driven framework for the systematic treatment of compliance requirements in software systems.

1.2 Problem Statement

Compliance management is an active research area, yet a systematic approach for evaluating and making value-driven decisions about compliance requirements is still an open problem. Typically, software engineers tend to use an informal, intuitive approach to compliance and our literature review shows that there is a lack of methods for analysing compliance requirements for software systems. As a result, it is difficult to identify and avoid compliance problems, as well as clearly and objectively describe how compliant a system is.

Stakeholders are interested in maximising returns of the systems while minimising risks and cost of development. The problem we address is how to evaluate compliance requirements in architectures quantitatively under uncertainty. The focus of this thesis is on using a value-driven approach to solve this problem. Compliance to industry best practices and standards can provide additional constraints and it is difficult to ensure that software systems comply with these constraints. To cope with the complexities of a value-driven approach to managing compliance requirements, there is a need for more research on a more comprehensive level that analyses how the different parts of the value creation process of requirement analysis process interrelate and on the economic nature of this relation. The research needs to take into account how to manage the risk, benefits, value and uncertainty during the development of the software and the resulting economic effect on the investment. The resulting solution approach should support traceability from low-level decisions to higher-level goals and compliance requirements that necessitate their existence.

These concerns call for a value-driven approach that takes into account the dynamics of the different value perspectives of stakeholders, associated risks, uncertainty and trade-offs. The main idea of our research is to design a framework for managing compliance based on value-driven approaches, to create value, maximise investments and visualise risk as a form of debt.

There is a lack of value-based systematic treatment of compliance requirements that can be applied across application domains and system designs. By treatment, we mean the application of methods that can aid in evaluating compliance requirements for software systems at design-time. Although the required compliance is set and governed by different regulations, it does not necessarily mean that all organisations must take the same route (processes and technologies) to achieving that compliance. These regulations can be adapted to the organisations to provide an optimum route and several sub-optimum routes based on different prioritisation factors such as risk appetite and cost. We argue that prioritising compliance requirements is a core activity to be performed when achieving compliance. We call for a generic systematic framework that is not domain specific and makes no assumptions on the system design. Proving that the method can be applied consistently across different domains is challenging in the lifetime of a PhD.

1.3 Thesis Overall Aim and Objectives

The aim of this work is to provide an economics-driven solution, which combines goal-oriented requirements engineering with portfolio-based thinking and technical debt analysis to enhance compliance related decisions at design-time while accounting for uncertainty. The approach is value driven, risk-aware, and systematic; it leverages on influential work of portfolio thinking to make the link between requirements, risks and technical debt explicit and transparent to software engineers. This approach allocates resources to achieving compliance requirements as well as looks at the associated technical debts, risk and value trade-offs. The main goal is to provide organisations with a set of guidelines to inform the decision of investment in compliance, based on economics, risks and technical debt.

In order to achieve this aim, this thesis seeks to answer the following research questions:

1. How can we systematically evaluate the compliance requirements in software systems under uncertainty, taking an economics-driven approach?
2. What are the requirements for such evaluation and how can we address these requirements?

Our research objectives can be concisely stated as:

1. To design a value-driven approach that is debt and value aware for managing compliance requirements during design time using the portfolio theory to optimise decisions and the concept of technical debt to provide more insights about compliance. This makes assumptions of uncertainty, limited resources for achieving compliance and that non-compliance may lead to incurring technical debt.

This thesis argues that financial valuation models and their underlying theories can be exploited to design a value-based approach for evaluating compliance requirements. This thesis studies the problem from a requirement and an architecture perspective. The primary motivation for this approach is to evaluate compliance at high level of abstraction that affords understanding the underpinning principles that makes one software more ”valuable” than another from the compliance point of view.

1.4 Overview of Proposed Solution and Thesis Contributions

To address the problems highlighted in the previous sections, this thesis adopts a value-driven software engineering perspective [30] and goal-oriented framework to evaluate compliance requirements for software-based systems. Traditionally, managing compliance requirements in software systems is primarily a technical process with minimal concern given to the economic perspective. Decisions are made based on technical values with little economic considerations. Regardless of how we define ”value”, [30] argues that engineering software is essentially an irreversible capital investment. Ensuring compliance, as part of the software development process, should add value to the software. Intrinsically, the costs of managing compliance requirements should not outweigh the returns from the software to achieve a net benefit.

The overall objective of this framework is to produce a value-driven method that is risk and value aware for managing compliance requirements at design-time. The outcome of this work will be beneficial to organisations because it proposes a novel method for ensuring better compliance management by viewing compliance as an investment activity that can maximise value. The framework offers several benefits to requirements engineers. First, it provides a systematic process description and some heuristics that can be applied while evaluating compliance requirements. Second, the emphasis that we place on the alignment of low-level architectural strategies with high-level goals and obstacle models of an application can help ensure that analysts can establish traceability among the different layers of abstractions. In addition, the debt metric produced while using the framework can be used to visualise value and risk, during the early phases of software development. The thesis makes the following specific contributions:

1. A novel economic- driven model that exploits portfolio theory to compliance requirements at the architectural level based on portfolio theory. The model builds on portfolio thinking for the valuation and provides a basis for investment decisions related to optimising software architectures. The model provides insights into allocating available resources in a manner, which maximizes value.

2. A novel approach that devises a model that exploits the concept of technical debt as a metric for visualising trade-offs between short-term and long-term value in compliance. This thesis introduces a new dimension of technical debt for explicating compliance debt at the architecture-level. The selection of an architecture under the constraint of compliance requirements is a typical multi-criteria decision making problem. In order to select a suitable architecture, alternatives have to be assessed against the requirements of the ranked according to their satisfaction to requirements. We present a well-defined approach to measure the satisfaction degree of requirements using a debt metric as a way to objectively inform the decision process.

3. This thesis builds on sound and influential work in architectural evaluation to propose a systematic five-phase method, which combines Goal Oriented Requirement Engineering (GORE) [117], the portfolio model and technical debt model for addressing compliance at design-time. This method is inspired by the Cost Benefit Analysis Method (CBAM) [98], allowing one to identify and quantify compliance in an architecture using utilities to quantify the satisfaction of the compliance requirement with specific architectural strategy and the overall system, respectively. By evaluating the architectural portfolio, it is possible to evaluate and compare quantitatively how compliant the architecture is.

4. The development of software systems involves a number of risks. In order to make rational decisions, stakeholders have to assess the risks involved in alternatives and balance them against the benefits of each alternative. Specifically, this thesis links

the concept of obstacles in GORE to debt, which is indicative of risk.

5. Background and Literature review:

   (a) An introduction to Goal-oriented requirement engineering and a representative sample of research works that have been done in the field of risk analysis using GORE.

   (b) An introduction to value-based software engineering

   (c) A conceptual analysis of compliance requirements that included a definition of compliance that is not dependent on the application domain.

   (d) Highlights the requirements for evaluating compliance requirements from an economics-driven perspective.

   (e) Technical Debt and Managing Technical Debt: This review presents the available definitions of technical debt in different fields and on different levels in the literature. The review also presents the available approaches, which have been investigating technical debt. It provides a comparison of different approaches dealing with technical debt on different levels, dimensions, causes, solutions, and evaluation methods.

   (f) Portfolio theory: This review provides a background on portfolio theory and an overview of closely related work on the use of portfolio theory in software engineering, and IT investments.

6. We evaluate the proposed framework on two non-trivial case studies to the applicability of the concepts, models and method described in this thesis.

1.5 Research Methodology

Scientific research calls for a disciplined approach to arrive at new findings or combine known results in novel ways. Research approaches vary depending on the investigated questions and research area involved. Broadly, Software Engineering researchers often make use of qualitative approaches (e.g. argumentation and case studies) or quantitative approaches (e.g. proofs and statistically deduced evidences) to validate results. Software architecture research, in particular, inherently produce results in the form of abstractions and processes that help to design high quality software systems.

The research methodology followed in this thesis constitutes five main iterative steps; problem definition, literature review, solution design, validation and assessment. A clear vision of the problem was established with the related research questions. The investigated research questions are initially validated via reviews to identify gaps in the literature. This provides a better understanding of the defined problem, better determination of the scope of the research. The thesis thereafter investigated the hypothesis by designing a preliminary solution to solve the defined problem and open research challenges considered within the scope of the research. This step involved the development of the portfolio model in chapter 3, the compliance debt valuation in chapter 4, and the method for evaluating compliance presented in chapter 5. These solutions provide efficient mechanisms to address the research questions highlighted in chapter 1. The solution proposed in this research was validated using case studies. Finally, we presented an assessment of the solution and limitations of the proposed solutions in chapter 7.

1.6 Structure of the Thesis

1. Chapter 1 has described the motivation and scope of the thesis. It included the research problem and hypothesis and the thesis contributions.

2. Chapter 2 covers the background concepts required to understand our work, such as experimental program analysis and goal-oriented requirements engineering. Our position with respect to related work is discussed. It introduces the concepts of goal-oriented requirement engineering and value-based software engineering, two important concepts that form part of the theoretical foundation for this thesis. Next, a conceptual analysis of compliance and compliance requirements is presented. Finally, this chapter presents a survey of the landscape of compliance requirements

to understand the state of the art and identify open problems

3. Chapter 3 presents an economics-driven model that exploits portfolio theory to address some of these requirements. The model provides insights into allocating available resources in a manner, which will minimize costs, reduce risk and maximize value. The model builds on portfolio thinking for the valuation and provides a basis for investment decisions.

4. Chapter 4 presents an economics-driven model that exploits the concept of technical debt as a metric for visualising trade-offs between short-term and long-term value in compliance.

5. Chapter 5 presents a method for evaluating compliance that combines techniques that have been developed throughout the course of this research.

6. Chapter 6 evaluates our techniques using case studies

7. Chapter 7 concludes the thesis and presents the future work

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

"The great thing about science is that you can get it wrong over and over again because what you're after - call it truth or understanding - waits patiently for you. Ultimately, you'll find the answer because it doesn't change."

Dudley Herschbach

This chapter reviews the concepts of goal-oriented requirement engineering and value-based software engineering. Next, a conceptual analysis of compliance and compliance requirements is presented. Finally, this chapter presents a survey of the landscape of compliance requirements to understand and identify open problems.

2.1 Preliminaries

Definition of the terminology is important because this thesis spans across multiple domains. Below we introduce an overview of those critical definitions to which we will refer throughout this dissertation.

1. **Compliance**: Compliance refers to an operating in agreement with established laws, regulations, standards, and specifications [102]. Essentially, compliance is a state of being compliant with a specific requirement, which means behaving how the requirement prescribes.

2. **Architecture**: A software architecture is the high-level structure and organisation of a software system. An architecture is a set of design decisions that intends to meet all of the functional requirements, while optimising desired quality attributes [174].

3. **Goal Oriented Requirement engineering**: Goal-orientation is an approach for managing requirements by representing requirements in form of goals

4. **Goals**: Goals capture the objectives to be satisfied; they can range from high-level business objectives, to well-defined technical properties [117].

5. **Obstacles**: While goals capture the objectives to be satisfied, obstacles capture undesired properties that may prevent the goal from being satisfied [120]. An obstacle obstructs a goal if the obstacle negates the goal in the domain [120].

6. **Technical Debt**: Brown et al. opined that ”like financial debt, technical debt incurs interest payments in the form of increased future costs owing to earlier quick and dirty design and implementation choices” [49]. The term technical debt has been developed broadly and has covered wider aspects associated with the overall systems development life cycle [49].

7. **Value-driven requirements Engineering**: Value-driven approaches reason why certain requirements are more desirable than others, and attempts to quantify the undesirable consequences of not achieving the requirement.

8. **Portfolio theory**: The goal of portfolio theory is to select the combination of assets using a formal mathematical procedure that can minimise risk for an expected level of return on investment while accounting for uncertainty of the real world. In finance, a portfolio denotes a collection of weighed compositions of assets (investments) by an investor, usually used as a strategy for minimising risk and maximising returns.

2.2 Goal Oriented Requirement Engineering

Goal orientation is set of techniques for elaborating, modelling and analysing software requirements [119]. In GORE, requirements are represented in form of goals. There are several definitions of goals in RE literature. Goals have been defined differently in different context and adapted to suit different objectives. A goal can be defined as an objective that a system under consideration should accomplish [119]. In defining a goal , Zave et al [187] states, "Goal formulation refers to intended properties to be achieved, they are optative statements as opposed to indicative ones, and bounded by the subject matter" [187] . In the case of regulatory compliance, the requirements are open-ended and they need to be defined with respect to a specific domain to add value.

In a detailed study of Goal-Oriented Requirement Engineering, Van Lamsweerde [119] defined a goal as an objective or a "*statement of intent that a system should satisfy*" [119]. Goals range from high-level business objectives to well-defined properties, which can cover compliance in our context. Agents are components, which are capable of performing operations to satisfy a goal [71]. In requirement engineering, goal driven approaches focus on why the system is needed, expressing the justification for a specific requirement. GORE approaches normally have the following activities: goal elicitation, goal refinement and various type of goal analysis, and finally the assignment of goals to the agents such as humans, devices and software. Several areas of research has been done to cover areas such as goal modelling, analysis and specification. The research has also been extended using goal-based reasoning for multiple problems such as trade-off analysis [2]; [3], risk modelling [8] etc.

There are number of benefits associated with modelling requirements as goals [117].

1. Goals provide an easy way to talk about requirements with the users. Goal refinement offers the right level of abstraction for involving decision makers to validate choices to be made between different alternatives and for proposing other alternatives [120].

2. Requirement Completeness – Goals ensure requirement completeness if all the goals related to a requirements can be proven to be achieved from the specification and the properties known about the domain considered.[122]; [120]; [117] .

3. Traceability - Goal trees show traceable relationships from high-level objectives to low-level technical requirements which can be assigned to agents ([117]; [120]; [122].

4. Conflict management of viewpoints- Goals can provide the basis for identification and management of conflicts between requirements [117]; [179].

5. GORE addresses the early requirements engineering phase during which stakeholders' goals are acquired and different options available for achieving these goals are explored [117] [122]. Finding the alternative system solutions is at the heart of GORE approaches [117].

6. Goals are usually more stable than the requirements as requirements do evolve [117]

7. Goals provide criteria for evaluating requirements relevance, to help avoid irrelevant requirements [117].

The Knowledge Acquisition in Automated Specification or Keep All Objects Satisfied (KAOS) methodology [179]; [68] is an approach in Goal-Oriented RE that aims at supporting the whole process of requirements elaboration and modelling. The method consists of identifying and refining high-level goals into subgoals. A set of subgoals are refined from parent goal if the satisfaction of all subgoals is sufficient to satisfy the parent goal. This refinement process continues until subgoals can be assigned to single agents. In KAOS, goals can be specified using natural language or defined formally using real-time temporal logic formalism. KAOS has been applied in requirements engineering problems such as supporting the management of conflicts between goals [179] , and the detection and resolution of obstacles that violate the achievement of goals [179] [120]. For a detailed survey on Goal-driven modelling, refer to [177] [9]

2.2.1 GORE for Risk Analysis

Goal-oriented Requirements Engineering approaaches have been used for risk analysis. In KAOS, the concepts of obstacles [120] and anti-goal [118] were introduced to capture undesirable conditions and the failure condition within a system design. An obstacle hinders a goal, and can lead to the goal not being satisfied. Anti-goals are goals, which are associated with malicious stakeholders such as a hacker. Obstacles can be considered as unintended risks, while anti-goals are threats or intended risks [12].

Ojameruaye and Bahsoon [25] proposed goal-oriented method and obstacles handling with portfolio-based thinking to systematically managing obstacles and refining compliance goals. While this paper presents an economic-technique for risk reduction, the goal-obstacle analysis is not adequate. TROPOS [45] is a qualitative goal-oriented framework that models soft goals and reasons about their contribution. KAOS goal models are extended in [54] with a probabilistic framework for obstacle analysis. According to Cailliau et al, obstacle analysis is a goal-oriented form of risk analysis. In [54] authors presented a probabilistic framework for obstacle assessment and goal specification. The probability and the seriousness of the outcome of obstacle occurrence is calculated by up-propagation from the leaf level goals. The computed information can be used to prioritize obstacles for obstacle resolution selection.

Asnar et al. [12] presented a framework for modelling and analysing the risk during RE phase was presented. The framework adopted the Tropos goal-modelling framework and proposed a qualitative reasoning process to analyse risks during the evaluation and selection between different options.

GORE has also been applied to multi-objective optimization decision problems under uncertainty [121]. Uncertainty makes decisions difficult and may result in significant risks. The authors have proposed to apply multi-objective optimization techniques and decision analysis to reason about uncertainty and its effect on the stakeholders' goals as well. They modelled the consequences of uncertainty using Monte Carlo simulations and shortlisted architectures based on expected costs, benefits and risks.

2.3 Value-based Software engineering

Value-Based Software Engineering (VBSE) is a discipline that integrates economic aspects and value considerations into software engineering principles and practices, processes, activities and tasks, technology, management and tools decisions in the software development context [30]. Formative studies on VBSE argue that value-neutral software development is responsible for project failures [156], [30]; [32]. Value-neutral software engineering emphasizes on technical activities while VBSE consider management oriented activities as part of the software lifecycle [30] [32]. Value-neutral approaches fail to recognise that the value of a software system gradually evolves with organizational and human needs [30]; [32]; [34].

VBSE is different from software engineering because it involves the concept of value [30]. Creating value is an economic activity that has to be taken into account during software development from a business perspective [156]. Value can be defined as ”relative worth, utility or importance” of an object compared to other objects [128]. In the Value-Based Software Engineering community, value is not only limited to purely financial terms, but extended to as relative worth, utility or importance to provide help address software engineering decisions[30]. In the context of software development, by borrowing the ideas from economic theory, Aurum et al. [156] identified the following fundamental aspects of value in software development:

1. Product value: This is the market value of the product and related to the product, and is influenced by the quality attributes of the software product[21]; [156].

2. Customer's perceived value: This is a measure of how much a customer is willing to pay for a product [156]. The customer's perceived value is driven by product benefit, price, and fulfilment of needs [50]; [156].

3. Relationship value: This value is derived from social relationships between the software creator and the user. The relationship value is driven by time, effort, product

development cost, price, time-to-market, differential advantage and customer satisfaction [156]; [50]

VBSE includes seven key elements: benefits realization analysis; stakeholder Win-Win negotiation; business case analysis; continuous risk and opportunity management; concurrent system and software engineering; value-based monitoring and control and finally, change as opportunity [30] . The engine at the centre is the Success-Critical Stakeholder (SCS) Win-Win Theory-W [31]. This addresses what values are important and how success is assured. It draws on four supporting theories:

1. Utility theory – How important are the values?
2. Decision theory – How does stakeholder's values affect decisions?
3. Dependency theory – How do dependencies affect value realization?
4. Control theory – How to adapt to change and optimise value realization?

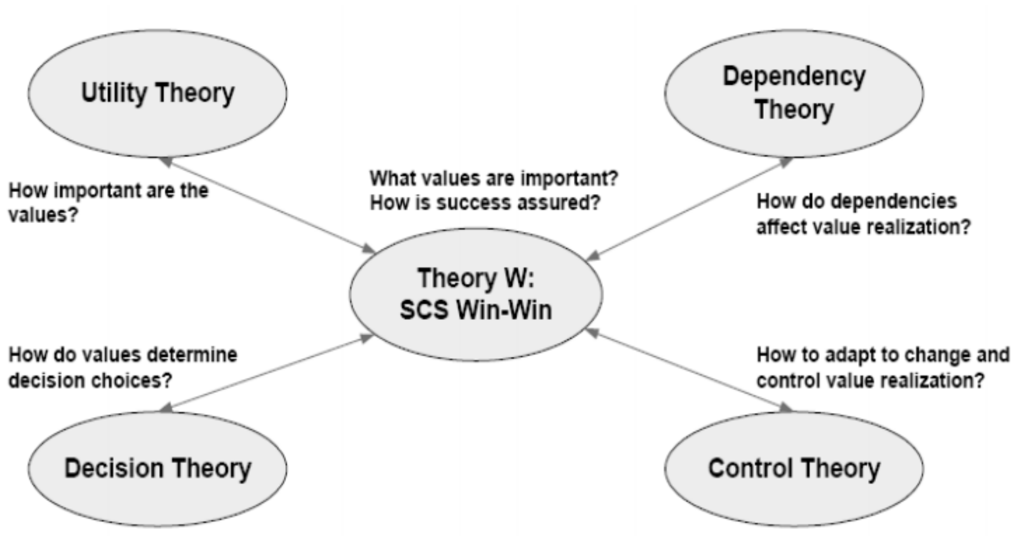

Utility Theory
Dependency Theory
How important are the values?
What values are important? How is success assured?
How do dependencies affect value realization?
Theory W: SCS Win-Win
How do values determine decision choices?
How to adapt to change and control value realization?
Decision Theory
Control Theory

Figure 2.1: The "4+1" Theory of VBSE [33]

2.3.1 Value-based Requirement Engineering

Value-based requirements engineering exploits the concept of economic value during the requirements engineering process. It is concerned with aligning requirements and business decisions for the creation of value [21]. In requirements selection decision, the stakeholder's value proposition and the risks associated with the requirements need to be considered for product value creation.

Quality requirements elicited from stakeholders tend to be idealistic, ambiguous and qualitative [86] . Consequently, these requirements are difficult to verify and either may not meet the stakeholders' expectations or may exceed their expectations resulting in either a useless system or an expensive system [156] [100]. Although, there are methods that predict development cost and benefits of complete systems [95], there is also a need to estimate or predict return-on-investment of individual requirements [78]. There is a need to model requirement dependencies [56] as the realisation of one requirement may have a negative impact on the value (cost-benefit) of another requirement. To summarize, some of the gaps in value based requirement engineering that are related to our research include

1. Value-based criteria for requirements selection
2. Objective estimation and prediction of risk factors to do cost-benefit analysis of individual requirements

2.3.2 Value-based Architecture

Value-based architecting involves the satisfaction of the system requirements with achievable architectural solutions [30]. A system architecture is the first design artefact that addresses the quality goals of the system [19]. Architecture evaluation is an activity for assessing an architecture against the quality goals of a system. A number of studies have been conducted in this area.

The Architecture Trade-off Analysis Method [108] (ATAM) does not only reveal how well an architecture satisfies particular quality goals, it also provides insight into how these goals interact with each other – how they trade off against each other [59]. ATAM is a scenario based architecture evaluation method. The Software Architecture Analysis Method (SAAM) [105] elicits stakeholder's input to identify explicitly the quality goals that the architecture is intended to satisfy. Unlike the ATAM, which operates around a broad collection of quality attributes, the SAAM concentrates on attributes for modifiability, variability (suitable for product line), and achievement of functionality. The Cost Benefit Analysis Method (CBAM) [106] [98] is a method for analysing the costs, benefits, and schedule implications of architectural decisions. The CBAM builds upon the ATAM to model the costs and benefits of architectural design decisions and to provide means of optimizing such decisions. Some approaches have considered identifying the optimal candidate architecture. GuideArch [74] guides architectural decision making such as ranking of the architectures and finding the optimal, under uncertainty using fuzzy logic. Letier [121] presents an evaluation method that allows describing effect of uncertainty on alternatives for realising stakeholders' goals. The method evaluates the consequences of uncertainty through Monte-Carlo simulation and shortlists candidate architectures based on expected costs, benefits and risks. They also assessed the value of obtaining additional information before making a decision.

2.3.3 Why a Value-Based Approach to Compliance ?

Generally, requirements engineering focuses on the system requirements. For a more balance view, it is also important to elicit the business goals related to the system requirements. This has been reflected in goal oriented requirement engineering. Traditional methodologies usually treat all aspects of software as equally important [30]. This leads to a purely technical evaluation that leaves the close relationship between compliance and business decisions unlinked and the potential value unexploited.

Quality requirements have been suggested by software engineers as the solution to many of the crucial challenges ranging from technical concerns to strategic concerns [77].

Favaro [77] emphasised the need to adopt management approaches that incorporate quality with a strategic framework. Value Based Management (VBM) includes a set of principles and processes that link quality related aspects to economic value, highlighting the unavoidable trade-offs between improvements in product quality versus higher economic cost, ROI versus market share, short-term gains versus competition [77].

We argue that compliance requirements are a concern and a constraint that pertain to quality requirements. A constraint is a requirement that limits the solution space beyond for meeting functional, performance, and specific quality requirements [86] . When compliance requirements are elicited, they are often stated in a way that is ambiguous and difficult to verify. Consequently, we may encounter the following problems:

1. Developing a system that is under designed and delivers less than is needed will result in a system that is useless as it is not compliant to regulatory requirements relevant to the domain.

2. Developing a system that is over-architected and delivers more than is needed. This results in an expensive system.

We take a similar approach to Favoro [78] [77] and argue that there is a need to align engineering compliance requirements with customer value, business value, product value and other relevant value dimensions quantitatively. We have described the link between value and compliance in chapter 3 and 4.

2.4 A Conceptual Analysis of Compliance in Software

Compliance has been studied from a range of application domains including the medical, economic, financial and technological domains. In the software engineering literature, there is a lack of uniformity in defining and measuring of compliance. This section analyses of the concept of compliance.

Maintaining compliance is an increasing area of concern for businesses. Compliance is a critical concern for systems and it is becoming more important in technology. Compliance can be used to describe organisational, system, process or user's behaviour. Most of the research on compliance from a requirements perspective has been concerned with the considerable challenge of eliciting and understanding compliance requirements. In an attempt to answer the question, “How does software engineering employ the term compliance?” relevant literature was reviewed and a critical analysis of this concept was undertaken. This included an examination of how compliance was defined and how it is related to other system goals.

Compliance deals with ensuring conformance with a legislations or guidelines. Compliance management deals with modelling, analysis and enforcement of compliance requirements. According to AMR research [17], there are three types of compliance companies try to address: regulatory, commercial and organisational. Works on regulatory compliance have been surveyed from several angles. Norris Syed Abdullah [1], presented an empirical study that examined the challenges in managing regulatory compliance from an information systems background.

Essentially, compliance is a state of being compliant with a specific requirement, which means behaving how the requirement prescribes. Arriving at a definitive definition for compliance from a value perspective is difficult as there are many different views on this. Compliance, like other quality concerns is not an absolute concept. It is relative to a set of business, system and quality goals as well as the application domain. To value compliance, one must specify with respect to which goals the system is compliant. Studies on compliance requirements make the implicit assumption that compliance equates to complying with legal rule. This does little to enhance an understanding of compliance requirements within organisational contexts. In order to understand more on the subject of compliance, we resorted to the definitions currently used in the existing literature. Among them, a work on compliance that was authored by *Julien Brunel* et al [51]. They capture common perceptions on compliance in this definition:

"A system is said to be compliant with a policy if either there are no violations or each time a violation occurs the associated sanction is enforced. "

Another definition by *Ralph Foorthis* [81] is:

"Compliance is a state of accordance between an actor's behaviour and product on the one side, and the predefined explicit rules, procedures, conventions, standards, guidelines, principles, legislation or other norms on the other."

Compliance is usually only realised and verifiable at run time The notion of international compliance is introduced by Siena et l [164] [165], "as the design time distribution of responsibilities such that, if every actor fulfils its goals, then actual compliance is ensured". The aim of intentional compliance is to make the top-level decisions for agents to be compliant. In terms of GORE, intentional compliance consists in identifying the set of goals such that, once satisfied in the design, compliance can be assumed[164]. In term of the system's architecture, intentional compliance consists of identifying a set of architectural strategies that once implemented, allow for arguing that a system is compliant. After extensive literature review, we settled on the following definition of compliance in requirement engineering:

Compliance is the ability of a system to satisfy its functional and quality goals to levels that are acceptable to predefined standards, guidelines, principles, legislation or other norms within the application domain. Compliance incorporates notions of cost, utility, trade-offs and management of risks.

2.4.1 Compliance and Compliance Requirements

Compliance requirements was defined by Kharbili [109] as a "piece of text extracted from a regulation that specifies a given regulatory guideline". In this thesis, compliance and compliance requirements are seen as meta-qualities or concerns – a requirement that relates to other system requirements. Therefore, a compliance requirement analysis evaluates the degree of a specific quality goal satisfaction in the context of the characteristics of the application domain and/or system design. For this reason, we evaluate compliance requirements within the context of quality requirements.

In general, compliance requirements can be classified into functional and non-functional properties [143]. Functional properties are concerned with verifying whether the system delivers what it is expected to deliver. Non-functional requirements are concerned with "quality" constraints such as availability, security, performance, etc. Compliance requirements can fall into any these two categories. For example, a compliance requirement that falls in the functional category may be "The user is immediately notified whenever her personnel data is collected". On the other hand, a compliance requirement that concerns a quality-of-service constraint may be "The average response time after submitting a loan request should not exceed 30 seconds".

Unlike conventional software requirements, compliance requirements have fundamentally distinct characteristics. These characteristics have been identified from the literature and summarised as follows:

1. Compliance requirements apply to multiple industries and application domains [38]

2. Compliance requirements are not elicited from stakeholders, they are elicited from regulatory documents [38]

3. Compliance requirements are ambiguous and can only be interpreted within the context of the application domain and system design. Their interpretation, prioritization and evaluation may vary according to the software application and domain.

4. Compliance requirements are intangible and depend on the functionality of the system. What a system needs to do and the expected compliance to a quality is a common starting point in software development. Despite the importance of the software being compliant, the question depends on specifying what the system need to comply with. This may lead to implicit or vague specification as well as bad evaluation.

5. Compliance requirements are subjective and can be evaluated differently by different stakeholders.

6. Compliance requirements are usually interconnected with other system requirements, satisfying one compliance requirement can affect the achievement of another requirement positively or negatively.

7. Compliance requirements are complex, they may be realised at a high level by a number of functions or a number of features at the lower level (components). Compliance requirements cannot be assigned directly to individual system components [165]. They need to be considered at the architecture or design of the system before they can be traced to system components. Another form of complexity is that compliance requirements require domain knowledge and the involvement of experts.

8. Evaluating compliance requirements is difficult as compliance is a run-time concern. Satisfying compliance requirements and quantitatively evaluating them require consideration of run-time factors such as different risk conditions. Therefore, compliance requirements might not be absolutely achieved; they may simply be satisfied sufficiently. Furthermore, quantitative evaluation can only measure some aspects of compliance requirements under specific conditions, which it is difficult to generalize for all system circumstances at run-time.

Evaluating and managing a system's compliance aim to assess the extent to which a system is compliant to specific requirements. Approaches to compliance management can be addressed at design-time.

2.4.2 Compliance at Design-time

Compliance driven design aims at ensuring compliance while a system is being designed. It is concerned with the static analysis of the system design before the its actual use against an applicable set of compliance requirements. This phase is highly iterative, it involves the continuous verification and modification of the system's architecture until an architecture that is compliant is achieved. Research in design-time compliance management is conducted in several directions. Some research efforts are geared to analyse, refine and manage high-level compliance requirements, which are abstract and ambiguous.

In design-time compliance evaluation, an architecture is considered as compliant with the set of relevant compliance requirements if it allows only for the operationalisation of requirements in instances not violating these compliance constraints [148]. Consequently, it can hypothetically guarantee that corresponding architecture is compliant. Managing compliance at design-time supports the idea of detecting and resolving any violation as early as possible [148]. This is a predictive approach to compliance and the evaluation is generally cost effective compared to run-time or retrospective approaches. This thesis examines a set of alternatives for operationalising (or satisfying) the compliance requirements. This begs the question: How can we ensure compliance in the system architecture? We pursue architectural strategies as a possible solution to describe these alternatives. To link the compliance requirements to the architectural strategies, we adopt Goal-Oriented Requirement Engineering where the goals are extracted from compliance documents. We then use a value-based approach to evaluate the extent to which the architecture satisfies the compliance requirements.

2.4.3 Compliance and System Goals

The analysis, evaluation and implementation of compliance requirements should be influenced by the application domain and system's design. Compliance is not only about the system state, but also about the system's ability to satisfy related quality goals with the system's application domain. The difference in the application domain characteristic determines the extent to which the system should satisfy the compliance requirements. The system's ability to satisfy these compliance requirements depends on its design, which may be achieved by varying the features of the system design. Determining the "best" way to satisfy compliance requirements, the critical system qualities that must be measured in order to judge compliance, and what constitutes satisfactory values for these system qualities, are all things that stakeholders ultimately must decide in the context of the system goals.

Compliance cuts across requirements and system design decisions. It refers to different system qualities, such as performance, availability, privacy and security. Compliance is therefore a meta-quality of other system goals. Consequently, it is vague to refer simply to “the system compliance”; instead, one must refer to the compliance with respect to a specific system quality, such as “the compliance with respect to security”, or “the compliance with respect to data privacy”.

2.4.4 Criteria for Evaluating Compliance

If one of the business goals of a system is that it should be compliant to a specific regulation and support value creation, it becomes necessary to evaluate how the compliance requirements are satisfied in the architecture. The evaluation has to relate technical issues to value. The evaluation should proactively address the economic ramifications of these compliance requirements and their impact on the system. Below, we highlight the concerns that should be addressed when analysing systems for compliance requirements at design-time.

1. **Intentionality:** Engineering and evaluating compliance requirements should be such that the system, once designed and implemented, will behave in the state intended by the compliance document [164] [165] .

2. **Ability**: While compliance requirements govern different domains, these requirements are operationalised by specific actors or agents. In general, any solution is acceptable only if the agent operationalising the compliance requirement has the capability to perform the necessary actions to satisfy that requirement [165].

3. **Traceability**: Compliance decisions are relevant in two different moments: when the decision is actually made, in order to align requirements with law and when the motivation of the requirements is needed [164]. The ability to trace compliance decisions to the relevant guidelines is essential as it supports the need for knowing which requirements are affected when regulatory guidelines changes. This traceability information is also the first element of proof to support compliance choices.

4. **Strategic Considerations**: A strategy refers to practises that treat uncertainty, incomplete knowledge, risk, cost, and related issues systematically, with the aim of maximizing the expected value of a given product or project [172]. The focus of strategic considerations is to improve the system decisions in meeting both its technical and business goals. The architecture is considered as the appropriate level of abstraction at which to think of strategic software decisions [18] such as compliance requirements. In this context, evaluating compliance requirements at the architectural level for value should address the following strategic viewpoint: (i) the cost of accommodating the compliance requirements with the architectural strategy (ii) the value implications with respect to risk and benefits of the architectural strategy in accommodating the compliance requirement.

5. **Addressing Uncertainty**: Uncertainty is defined as an event that can happen, but the probability of its occurrence is unknown [96]. Uncertainty can be associated with changes in compliance requirement, its associated risk's complexity and likelihood. Changes in compliance requirements could be considered a major source of uncertainty that may affect any investment in a system. Uncertainty can also be associated with risks associated with the way the compliance requirements are satisfied.

2.4.5 Linking Compliance Requirements to Architecture

The vast majority of works we reviewed in the area of compliance analysis and management are concerned with elicitation, modelling and metrics that estimate the extent to which a compliance requirement is implementation ready. These works tend to view compliance and compliance requirements as "stand-alone" quality requirements. However, we view compliance as a quality that relates to other system qualities and has a significant impact on the architecture. Hence, the work presented in this thesis also relates to the area of architecture.

Value-based solutions for linking requirements engineering activities to architecture have been discussed by [107]; [89]. This is particularly important when considering compliance requirements in value-based context. Carrying out the requirements negotiation and architecture evaluation as separate concerns, it can lead to unnecessary iterations since many requirements are not discovered or clarified until the architecture is being designed. This would consequently result in wasted effort, re-work and extra cost later in the development cycle.

Some methods concerned with the analysing software qualities at the architectural level include : The Scenario-based Architecture Analysis Method (SAAM) [105], Architecture Trade-off Analysis Method (ATAM) [108], and the Cost Benefit Analysis Method (CBAM) [106]. These methods generally identify the quality goals of interest and then evaluate the strengths and weaknesses of the architecture to meet the desired goals. Depending on the method, the evaluation explicitly addresses a single quality or multiple quality goals at the same time.

Grunbacher et al. [89] have proposed a solution called CBSP, a lightweight approach intended to provide a systematic way of reconciling requirements and architectures using intermediate models. CBSP uses a simple set of architectural strategies to refine the requirement into an intermediate model facilitating their mapping to architectures. CBSP addresses the issue of differences in concepts and terminologies between requirements engineering and architecture. However, linking of business values and architecture quality attributes needs to be explicit.

2.4.6 Compliance and Multiple Criteria Optimisation

In this thesis, achieving compliance is seen as a multi-criteria optimisation problem, where multiple and sometimes conflicting compliance requirements have to be satisfied simultaneously.

Analysing compliance requirements at the architectural level evaluates the extent to which related quality goals are satisfied within the application domain. The compliance goals are quantified by utility over the value of each value perspective and related quality goals. A utility value therefore measures the “satisfaction level” of the compliance goal with respect to individual compliance dimensions and value perspectives. The utility value can be thought of as a measure of the “overall satisfaction” of the compliance requirement in terms of value.

2.5 Research Effort on Managing Compliance Requirements

Compliance requirements have been studied from different requirement engineering contexts, such as elicitation [41] [131] [132][164] [13] [85] [42], prioritisation [130] [129] , [164], value-based approaches [52] and tool support [40] [93]. The thesis followed a systematic review methodology as proposed by Kitchenham et al. [112] to investigate the state of the art in value-driven compliance requirements research. We also applied a concept-centric literature review method to classify and present the research articles according to the area they address [112]. The objective of this systematic literature review is to find publications that apply value-based methodologies, for managing compliance requirements. This thesis studies papers that have implications for the design of compliant software systems. Specially, the following research questions steered the review:

> What value-based frameworks are there to help organizations achieve and manage their legal compliance?

Accordingly, we define a set of secondary sub-questions as follows:

- Are there any methods for prioritising legal compliance requirements?
- Are there any value-based based methods or frameworks for managing legal compliance?
- What are the limitations of the current research?

Several requirement-engineering frameworks have been proposed for managing compliance. Breaux [39] proposed a distributed requirement management framework, which ensures that legal obligations are incorporated into software requirements. Although this framework provides details on analysing compliance, the approach does not integrate the concept of value. A framework on generating legally compliance requirements is proposed by Siena et al. [164] [140]. The process starts by defining the relevant laws, then a legal model based on the Hohfeldian taxonomy is developed. The output is a refinement of the model into intentional elements and goal models. This work fails to provide a way of integrating other requirements not related to compliance.

Semantic Parametrization is used to extract rights and obligations from regulations and to clarify their potential ambiguities [40] by generating a formal legal model by eliciting legal requirements in terms of permissions, obligations, and constraints from legal texts.

In requirements engineering, researchers have investigated different methods for analysing security requirements using goals, with more recent work focusing the extraction of requirements from security policies [43] [133]. The work of Anton and Breaux [42] takes this further by systematically extracting rights and obligations from legal texts.

Massey et al. [130] presents an approach for prioritising legal requirements that calculates a prioritization scale computed from mapping of requirements to legal documents. Requirements are triaged and then categorized into two groups: ready to implement or needs more evaluation. This approach does not consider the level of importance of the legal requirements from an value perspective. The the Frame-based Requirements Analysis Method (FBRAM) developed by [129] produces a comprehensive prioritization hierarchy used to identify which requirement should be prioritised from legal text [129].

In the area of value-driven compliance, this includes papers that describe compliance management frameworks that include value-based perspectives into managing compliance requirements. These types of frameworks have to integrate value or economic models with legal requirements into a framework and aim to achieve and maintain compliance. A value-based approach for managing compliance includes the principles for eliciting requirements with respect to value propositions, requirement analysis, prioritisation of desired compliance factors and mitigating the risk of non-compliance. There is little work done using value based approaches to manage compliance requirements. In this category, very few types of frameworks were identified.

One important contribution in this category Burgemeestree et al, [52] they discussed how value-based augmentation theory can be applied to formalising compliance decision. This approach models a control system and the justification for compliance decisions / choosing control in a state transition diagram. It operationalizes legislation into control objectives and identifies the control measures. This approach also takes into account the organisational context of the legislation. Although this approach helps to formalise compliance decisions, it does not present a holistic single value-based approach for managing compliance.

In the area of compliance in adaptive systems, [84] proposed a process to support adaptive compliance that extends the traditional compliance management lifecycle with the activities of the Monitor-Analyse-Plan-Execute (MAPE) loop, and achieves adaptation through reconfiguration They defined adaptive compliance as the capability of a software system to continue to satisfy its compliance requirements, even when run-time variability occurs. Using a case study of an industrial RE project, [139] identified a number of key impediments to achieving regulatory compliance.

Even with significant contributions in the field of compliance and requirements, there are still some research problems that deserve more attention. Value-driven management of compliance requirements and architectures have barely been explored. Compliance has a major influence on the organisation's cost and value and having a value-based framework that can help companies to underpin their compliance decisions and justify their value is important.It is necessary to consider that legal requirements do not all have the same impact on organizational objectives. Furthermore, not all non-compliance instances have the same importance with respect to law. Ensuring compliance is a time-consuming and expensive activity for organizations. Therefore, it is critical for them to be able to prioritize tasks related to compliance. This motivates the need for more research on value-driven methodologies for analysing compliance requirements.

2.6 Summary

Although there are substantial contributions in the field of compliance requirements, there are still some open research questions. It is crucial to note that compliance requirements do not all have the same impact on stakeholders' objectives. Their implementation will vary with the application's domain and system's design. In addition, not all instances of non-compliance have the same importance with respect to law. Ensuring compliance is a time-consuming and expensive activity. Therefore, it is critical for them to be able to prioritize tasks related to compliance. This motivates the need for more research on methodologies for prioritizing legal requirements. Compliance has a major influence on the system's cost and value and having a value-based framework that can help companies to underpin their compliance decisions and justify their value is important. In the treatment of compliance requirements, we identified three gaps:

1. Linking compliance from requirements to architectural decisions

2. Linking compliance to value

3. Visualising compliance trade-offs in architectures.

An examination of compliance requirements in software systems requires background knowledge in many fields. Section 2.2 presented a review on the relevant background concepts and related works. It introduced and described a review of Goal-Oriented Requirement Engineering. Having reviewed the literature on goal, we recommended the use of GORE for modelling compliance requirements as goals due to the associated benefits such as traceability and ease of use.

In section 2.3, value-based software engineering was introduced. Traditional methodologies usually treat all aspects of software as equally important [32] 2003). This leads to a purely technical issue leaving the close relationship between compliance and business decisions unlinked and the potential value contribution unexploited. This section presented the need to align engineering compliance requirements with customer value, business value and product value and other relevant value dimensions quantitatively.

Compliance has been studied in many contexts. Section 2.5 introduces the concept of compliance in software systems. It offered the definition of compliance and compliance requirements used in this thesis. It is important to note that we are not proposing a generic definition of compliance requirements. These definitions are used only as a reference to provide scope and consistency while explaining the proposed method. This section also looked at conceptualising compliance by linking compliance to system and quality goals. It differentiated between compliance at run-time and compliance at design time. This thesis specifically looks at compliance at design-time, which incorporates the early phases of system development. To address the challenge in dealing with compliance requirements, this section highlighted the criteria for evaluating compliance requirements from a value-driven software engineering perspective. The approach shall provide the basis for analysing many of the trade-offs involved in engineering for compliance at design-time.

Findings from this review provide evidence to support the claim that existing approaches offer limited primitives for evaluating compliance at design-time for value. Therefore, we motivate the need for a novel value-aware approach that addresses these limitations.

Our research pursues the goal of exploiting principles presented in the related works discussed in this chapter. The work presented in this thesis also relates to architectural evaluation. This research is concerned with analysing software requirement concerns such as compliance at the architectural level. The evaluation looks at the problem as a multi-criteria optimisation, and explicitly addresses a single or multiple attributes (goals) at the same time. The work also relates to the field of requirements engineering because goals used in the analysis are derived from compliance requirements. To the best of our knowledge, there is no work on analysing compliance requirements from a value perspective, and current risk analysis methods do not provide systematic guidance for visualising compliance risk in architectures.

CHAPTER 3

THE MODEL FOR EVALUATING COMPLIANCE REQUIREMENTS WITH PORTFOLIO THEORY

”All models are wrong, but some are useful.”

George E. P. Box

In the previous chapter, we highlighted the requirements for conceptualising and evaluating compliance requirements. In this chapter, we pursue an economics-driven model that exploits portfolio theory to address some of these requirements. The model provides insights into allocating available resources in a manner, which will minimize costs, reduce risk and maximize value. The model builds on portfolio thinking for the valuation and provides a basis for investment decisions.

We first provide background on portfolio theory that is necessary to understand our approach. We then describe the portfolio-based approach to the systematic evaluation of compliance requirements, leading to the model. We show how we have derived the model, the analogy and the assumptions that the model makes, the model formulation and its sensitivity. Finally, we provide an overview of closely related work on the use of portfolio theory in software design and engineering.

3.1 Portfolio Theory: A Brief Background

3.1.1 Definition

A portfolio denotes a collection of weighed compositions of assets (investments) by an investor, usually used as a strategy for minimising risk and maximising returns [25]. Portfolio theory attempts to show the benefits of holding a diversified portfolio of risky assets rather than assets selected individually. The theory can also assist in determining the optimal strategy for diversification of assets to minimise risk and maximise return [25] . Modern Portfolio theory [127] was introduced in 1952 by Harry Markowitz. The goal of modern portfolio theory is to select the combination of assets using a formal mathematical procedure that can minimise risk for an expected level of return on investment while accounting for uncertainty of the real world.

Portfolio optimisation aims at constructing portfolios that maximize expected returns consistent with individually acceptable levels of risk. Using both historical data and investor expectations of future returns, portfolio selection uses modelling techniques to quantify "expected portfolio returns" and "acceptable levels of portfolio risk," [150]. This theory supports the quantifications of investment risk and expected return of a portfolio, hence providing an objective approach to investment management [150]. Initially, the focus of portfolio management used to be the risk of individual assets; nonetheless, the theory of portfolio selection has shifted the focus to the risk of the entire portfolio. The theory shows that it is possible to combine risky assets and produce a portfolio whose expected return reflects its components, but with the potential for considerably lower risk [150].

3.1.2 What Problems Do Portfolios Address?

Portfolio theory is a branch of financial economics, which aims at minimizing the risk of an investment strategy for a given level of expected return by choosing different combination of assets [127] [166]; [72]. The two main variables for achieving the minimization of the risk is deciding which assets to invest in and the size of the investments in relationship to the total investment [144]. Depending on the return associated with different assets, it is possible to find a combination of these assets with a lower combined risk profile. Given the above, the underlying problem in portfolio theory is to find the weights and combinations of the different assets that minimize the risk in the portfolio. The theory can also assist in determining the optimal strategy for diversification of assets to minimise risk and maximise return [25].

Portfolio analysis has been extensively applied to various sectors such as information retrieval [182], energy [70] [10] [16] [14] [15], health economics [46], biodiversity [63] [79], real estate [83] [91], product development [61] , project management and IT investments [20] [181] [27]. The application of portfolio theory in software engineering is detailed in Section 3.4 of this chapter.

3.1.3 Portfolio Valuation

A vital decision regarding investment is the amount of risk an investor is willing to bear. Ideally, an investor will want the lowest possible risk, while obtaining the highest amount of return. Higher returns tend to be associated with larger risk, hence there needs to be a trade-off between risk and return. The ideal goal in portfolio management is to create an optimal portfolio derived from the best risk-return opportunities for a particular set of constraints [44]. The concept of risk is important to any discussion on portfolio management.

Risk is difficult to define and measure. The presence of risk in any investment means that more than one outcome is possible. An investment is expected to produce different returns depending on the set of conditions. The most common method for measuring risk in portfolio theory is the standard deviation of the expected returns. The standard deviation is a measure of risk, the greater the standard deviation, the greater the risk. It is a function of variance.

$$Standard\ Deviation,\ \sigma\ =\sqrt{variance} \tag{1}$$

One way to control portfolio risk is via diversification of assets to reduce the exposure to the risk without reducing the level of return. To measure the success of a potential portfolio, covariance and correlation are considered. Covariance measures to what degree the returns of two risky assets correlate. A positive covariance means that the returns of the two assets move together, while a negative covariance means that they move in opposite directions.

For two investments x and y, where p is the probability, the covariance

$$COV(x,y)\ =\sum p\ (x{-}x_1)(y{-}y_1) \tag{2}$$

The correlation coefficient r,

$$r\ =\frac{COVxy}{\sigma_x\sigma_y} \tag{3}$$

1. When r = +1 is a perfect positive correlation. This occurs when the returns from two assets move up and down together in proportion.

2. When r = -1 is a perfect negative correlation. This takes place when one asset moves up and the other one goes down in exact proportion. Combining these two assets in a portfolio would increase the diversification effect.

3. An uncorrelated (r = 0) occurs when returns from two assets move independently of each other. Combining these two assets in a portfolio may or may not create a diversification effect. However, this position is better than in a perfect positive correlation situation.

Given the following inputs – returns, standard deviations, and correlations – a minimum-variance portfolio for an expected return can be calculated. The return on a portfolio of assets over a specified period is the weighted average of the individual assets in the portfolio

$$Rp \ = w_1R_1 + \ w_2R_2 + \ldots \ W_nR_n \tag{4}$$

$$Rp \ = \sum_{n=1}^{n} WnRn \tag{5}$$

In general, for a portfolio with i assets, the portfolio risk is given by

$$R_P \ = \sqrt{\sum_{i=1}^{m} W_1^2 R_1^2 + \sum_{i=1}^{m} \quad \sum_{j=1}^{m} W_i W_j R_i R_j P_{ij}} \tag{6}$$

Portfolios that provide the biggest possible expected return for given levels of risk are called efficient portfolios [75]. To construct an efficient portfolio, some assumptions are made about investor behaviour when making investment decisions. A practical assumption is that investors are risk averse, and when faced with choosing between two investments with the same expected return but two different risks, they prefer the one with the lower risk [75]. In selecting portfolios, the goal is to maximize the expected portfolio return for a given tolerance for risk. Given a choice from the set of efficient portfolios, an optimal portfolio is the one that is most fulfils the investor's requirements. This preference is expressed using a utility function.

In order to construct efficient portfolios, the theory makes some basic assumptions about asset selection behaviour by investors. The assumptions are as follows:

1. The only two parameters that affect an investor's decision are the expected return and the risk.

2. Investors are risk averse [150].

3. All investors seek to achieve the highest expected return at a given level of risk [150].

4. All investors have the same expectations regarding expected return, variance, and covariance for all assets [150].

5. All investors have a common one-period investment horizon [150].

3.2 Compliance Requirements: A Portfolio Perspective

In the previous chapter, we highlighted the requirements for regulatory requirements. These requirements necessitate finding an approach, which assesses uncertainty and traces technical issues to value creation.

3.2.1 Economic Perspective

We approach the analysing compliance requirements problem from a value-based perspective [35] [32]. Economics-driven software engineering has acknowledged that integrating value-based theories with the technical decision making can yield measurable improvements in development cost, time and risk management [32].

Managing compliance to compliance requirements is ultimately an investment activity that requires value-driven decision-making – about selecting the right compliance goals and handling the obstacles to those goals for mitigating risks [25] . This need becomes more intense with compliance requirements where their value tends to be invisible, as they do not generate revenue although they are crucial to the business sustainability. The value of these requirements are usually questioned and the situation is aggravated in organisations that must balance very limited resources with requirements that have visible value chain [25].

3.2.2 A Motivating Case study

As a motivating example adapted from the literature [76], consider a Bank- referred to as Smart Bank throughout this thesis. Smart bank has recently decided to adopt a Cloud provider platform to deliver a robust, scalable and on-demand service provisioning for the risk management aspect of its business [76]. The following constraints must be strictly adhered to according to the problem definition:

1. According to the territorial laws of the country in which the bank operates, there

are policies that prohibits certain regulated data from being hosted on any servers outside the territory [Security]

2. Any mechanism adopted to secure the data in order to meet the constraint above should not affect performance significantly. [Performance]
3. Any component(s) added to achieve the systems's security goals should not be a single point of failure. [Availability, Security]

As an example, Smart bank has to comply with the information security regulatory requirements to ensure confidentiality, integrity and availability of information. Then bank is responsible for ensuring the implementation of the application considers and resolves risks. There is also the need to manage any risks as well as identify an optimal portfolios that will be implemented and quantify any resultant associated debt incurred.

Compliance to compliance requirements is a multi-dimensional problem that involves reasoning about risk, cost and benefits, and cannot be fully explained by individual technologies. Its multi-dimensional nature makes it difficult to reason about these compliance requirements and build compliant systems that accommodate all these relevant dimensions.

In general, this case can be modelled as a multi-objective optimisation problem which the customer's utility, the implementation costs and risks formed the three objectives. The optimal solutions identify the optimal implementation alternative and specify the proportions and value of the corresponding design variables. The question of interest, however, is how can we reason about and value different compliance requirements? How can we select and implement requirements, which minimises the risk and uncertainty of non-compliance? How does investing in implementing these compliance requirements result in reducing risk and maximising value?

3.2.3 Why a Portfolio Perspective?

Portfolio theory is well suited to address some software engineering problems from a value-based perspective [32]. To understand and analyse compliance requirements in software systems using an economic approach, we need a valuation technique that is suitable for strategic, short-term and long-term valuations, that accounts for uncertainty and minimize risks for a given level of expected investments while quantifying any incurred debt. Portfolio theory satisfies these requirements.

First, portfolio theory provides an analysis paradigm that emphasises the value of maximizing the expected return for a given amount of portfolio risk, or minimizing risk for a given level of expected return, by carefully choosing the proportions of various assets [127]. In traditional applications with the financial industry, portfolio analysis recognises that the value of investments lies not only for maximising the returns that the investment is expected to generate, but also in minimizing of the risk. A key question then becomes which functionalities this software system should include. The problem is that different software will have different needs of functionalities and different customers will have a different willingness to pay for different functionalities. From the developing firm's point of view, the question becomes what functionalities should be included in the software system and what functionalities should be developed for the individual software. In order to reduce the market uncertainty of the line of software that the firm is developing, or is expecting to develop, the firm wants to develop a system that is most usable for this strategy. From an investment point of view, it means that the firms should invest in software system that minimizes the market uncertainty to the greatest extent. The combinations of software functionalities that accomplish this will be denoted as efficient software systems. From an economic point of view, the firm wants to maximize the return of the investment while minimizing the risk. To do so the firm will invest different amount of money in different functionalities

Engineering software system requirements is characterized by risk and uncertainty [29] [28] where risk is an operationalized measure of the uncertainty. This uncertainty can be divided into two main issues; what shall be developed and how it will be developed. The former type of uncertainty is referred to as market uncertainty, i.e. uncertainty concerning the expected return or utility from the selected requirement. The latter type is referred to as technical uncertainty, i.e. uncertainty concerning the development cost of the requirement. The underlying problem in portfolio theory is to find the weights of the different assets that minimize the risk in the portfolio given the different assets return characteristics. A software system can be seen as a portfolio of different software requirements. The problem of selecting what and how to develop these requirements is therefore similar to the classic portfolio optimization problem in financial economics in a simplified setting in the way it will be analysed in this thesis. We can therefore use portfolio theory to analyse what constitutes an efficient combinations of architectural strategies to guide investment decisions in satisfying compliance requirements. An efficient system architecture is a combination of architectural strategies for satisfying requirements that yields a lower risk than other combinations for a given level of investment.

3.3 The Model: Valuing Compliance Requirements with a Portfolio Theory Analogy

Subsequent sections describe the portfolio theory-based approach for analysing compliance with an analogy using modern portfolio theory [127]. We describe the approach, present the analogy, formulate and interpret the portfolio model. Subsequently, we report on its sensitivity and on its possible uses. Finally, we discuss valuation issues and assumptions under the model.

3.3.1 The Approach

We assume that one of the business's goals is to comply with relevant compliance requirements. We view this goal as a value-seeking and value-maximising activity: requirement engineering is a process in which system requirements are elicited and specified. We attribute value to the ability of a software system in reducing risk and optimising cost of complying with compliance requirements. In this perspective, we rely on intuition in relating compliance to value: achieving compliance is a strategic resource that can be built in or adapted into a system with the objective of creating value.

The goal of the requirement engineering process is to create a detailed recipe of design decisions for developing a software that will satisfy functional, quality and business requirements. Consequently, achieving this goal necessitates active decision analysis of the goals and architectural strategies (AS) in order to ensure stakeholders' satisfaction and optimal investment decisions under uncertainty. A portfolio- driven approach involves decision-making aimed at creating a mix of requirements that returns maximum value for the resources invested. This leads to critical decisions involving the selection of an optimum mix of requirements aimed at satisfying the necessary goals.

A software system can be referred to as a diversified set of assets, a portfolio of requirements with diverse risks and benefits. Each type of requirement has different drivers of risks that may be correlated with one another. Those uncertainties and interactions must be taken into account when deciding over the composition of the portfolio of assets. Since projects usually have limited resources to invest, and given the diversity of the different sources of risks, as well as the diversity of requirements and possible portfolios, it is reasonable to use the concept of risk and return from portfolio theory to support decisions in requirement engineering. This looks at the architecture and portfolio selection as the trade-off between risk and return. Portfolio theory refers to risk as the dispersion of return.

It can be argued that software engineering has similarities to the financial investments. However, when portfolio theory is used to support the requirement engineering process, a few assumptions need to be taken into account:

1. The expected return E_1 of investing in a specific requirement is equal to the expected added value of the requirement improving compliance

2. We rely on expert judgement and relevant historical data if available, to predict likely future risk. This Risk R_1 is a function of obstacles and agent liability.

3. We assume that the analysts will complement the decision making process by evaluating and quantifying the extent to which requirements are correlated. This can

be done by modelling the requirement interdependencies. The correlation P_{ij} describes the direction and strength of the relationship between requirements R_1 and R_2 in terms of their impact on regulatory compliance parameters. The correlation is represented as a number between -1 (a perfectly negative correlation between the two items) and +1 (a perfectly positive correlation).

Successful use of portfolio thinking is about achieving a balance between these potentially conflicting goals

1. Maximising the value of the architectural portfolio
2. Linking architectural decisions to strategy
3. Optimising the decisions related to compliance requirements on relevant dimension

We claim that using a portfolio approach to compliance can add value to a software system. The value is attributed to reduced risk. The value in the context of compliance is strategic in nature and may not be visible immediately. It takes the form of savings through avoiding over-architecting and reducing risk through avoiding under-architecting for compliance. In software systems, uncertainty is a major source of risk that confronts the architecture during its lifetime. Minimising risk under has a value. The importance of the idea cannot be overemphasized as it provides an effective way to reason about a crucial but previously intangible source of value and to employ it in the evaluation of compliance.

We contribute an approach for evaluating compliance in software systems inspired by portfolio theory. As we have mentioned in an earlier chapter, approaches to evaluating compliance can be at design-time or run-time. We contribute a design-time approach, where we use value-based reasoning for evaluating compliance. Briefly, the approach considers the architecture as the appropriate level of abstraction at which to think about strategic investment decisions, guide the analysis of compliance requirements, and analyse the value, costs, and investment opportunities. The approach builds on a sound theory in financial engineering to provide “insights” into investment decisions, and a basis for analysis for optimising architectures for compliance requirements.

3.3.2 Fundamental hypothesis

Establishment of this model aims at introducing the concept of risk, making the risk and return the main variables to determine the selection decisions using the portfolio analysis. This realizes the optimal combination of maximum return and minimum risk. Thus, we make the hypothesis as follows

1. The stakeholders aim to maximise returns and evade risk, which means they would optimize the investment portfolio on the consideration of both expected return and risk.

The constraint conditions for the model

Suppose there are n requirements included in the portfolio, the weight for each X$_i$(i = 1, 2, 3 ... n) respectively, consequently X = [X$_1$, X$_2$ ... X$_n$]^r is the decision variable for investment allocation. The constraint condition presented in the portfolio model is:

1. Total investment constraint: The weight of each requirement sums up to 1, that is

$$\sum_{i=1}^{n} x = 1 \tag{7}$$

3.3.3 The Mathematical Model

Taking into account the assumptions highlighted in 3.1.2, we can apply the portfolio theory to the problem. Each requirement will have its own benefit, risk, expected return, cost and correlation with other requirements. Based on these values we can then decide how to construct an optimal portfolio so the risk is reduced.

The structure of software development is complex, as it includes functional and quality requirements as well as architectural decisions. From a development point of view, the value of a requirement is derived from its expected benefit to the customer and the product. According to financial theory, ensuring diversification can improve the risk/return ratio. Software is a typical diversifiable asset portfolio that includes a range of functional and quality requirements. These requirements also have obstacles that must be resolved. Considering the relative risk and constraint conditions in software development process, this thesis establishes the objective programming model to reach the optimal portfolio in requirements based on portfolio theory.

The expected benefit of the individual requirement is calculated based on selected value perspectives and impact parameters that are relevant to the domain. We assume that parameters A and B relevant to a domain and will be impacted by a compliance given requirement, while A_1 and B_1 represent quantitative values that measure the impact on a requirement on the respective attributes. Usually qualitative metrics may be extended by quantifying or replacing value such as high, medium or low with numerical weights.

The degree, to which each of these attributes is expected varies from application to application and the specific domains. This degree is measured by a priority weight W_A and W_B, which measures the importance of each attribute. The priority weight can be a number between zero and one, and this is assigned based on the needs of the organisation. However, the sum of the weights cannot exceed one.

The Expected Benefit B_1 for a R_1 is calculated as

$$B_1 = W_A A_1 + W_B B_1 \tag{8}$$

With one constraint represented in

$$W_A + W_B = 1 \tag{9}$$

The expected benefit B of any requirement R is

$$B_1 = \sum_{i=1}^{n} (PriorityWeight)(AttributeImpact) \tag{10}$$

We can model the expected return in requirements engineering using valuation techniques such as cost-benefit analysis and utility theory. The expected return on the requirement R with an associated cost C is

$$E_R = B/C \tag{11}$$

Consider X_i is the weight of each requirement in the software portfolio and B_i is the expected benefit for the *ith* requirement, the expected benefit E_P of the software portfolio can be calculated with the equation

$$E_P = \sum_{i=1}^{n} X_\mathrm{i} B_\mathrm{i} \tag{12}$$

Subject to the constraint

$$\sum_{i=1}^{n} X_\mathrm{i} = 1 \tag{13}$$

The risk of the portfolio is affected by the risk associated with each asset (requirement). This is explained further in chapter 5. The risk of the portfolio R_p is calculated as

$$R_P = \sqrt{\sum_{i=\mathrm{i}}^{m} W_\mathrm{i}^2 R_1^2 + \sum_{i=1}^{m} \quad \sum_{j=1}^{m} W_i W_j R_i R_j P_{ij}} \tag{14}$$

The correlation coefficient between the two ranges from -1 to 1, the risk of the portfolio increases gradually.

1. When the correlation P_{ij} is one, investment returns of different architectural strategies exhibit a positive correlation. The risk of portfolio will correspond to the weighted average of investment risk of different requirements in the portfolio when the risk of the portfolio is maximal.

2. When the correlation Pij is -1, the risk of the portfolio is minimal. Therefore, as long as investment returns of different architectural strategies do not exhibit perfectly positive correlation, the risk of software portfolio will always be lower than

the weighted average of risk to invest different requirements individually. The architectural portfolio can diminish risk while maximising returns against compliance requirements.

3.3.4 Portfolio Selection

The mapping of the risk-return sets of possible portfolios mentioned above is done considering the relative weight of each type of asset in the portfolio. However, this relative weight is an adequate reference to assess the portfolios. The cost of the investment should be accounted for when making portfolio decisions.

Markowitz [127], identified the that efficient frontier of risk and return for all portfolios that cannot be improved in both risk and return among the alternative options. From all possible portfolios of N assets, the selection must only consider the portfolios over the optimal Pareto frontier of the domain of options.

3.3.5 The Analogy

A major insight behind this portfolio model is that optimisation for compliance requirements is analogous to financial assets: investing in an optimum combination of architectural strategies as assets is analogous to investing in an optimised portfolio of assets. Having set the selection of requirements under risk, uncertainty and value as an optimisation problem, the challenge becomes valuing that selection. We build on a simple and intuitive analogy with portfolio thinking to value different portfolios of requirements. In this section, we formulate the model and explore the analogy the model makes with modern portfolio theory.

Let us assume that the potential or value (V) of a given system is a function of the benefits that can be derived from the system and the associated risks. The system is a collection of requirements with varying levels of investment in each requirements. These compliance requirements have associated benefits and risks, which in turns affects the expected benefit and risks of the system. It can be seen that a system as a portfolio has similarities to investment financial market with the view of diversification to reduce risks and maximise returns. However, we use portfolio theory to identify an architecture as optimal portfolio of architectural strategies with a total value that minimises risks and maximises return on a system.

The system's portfolio is derived by mapping the economic characteristics of the architecture onto the parameters of the portfolio model as shown in table 3.1.

Table 3.1: Financial portfolio theory analogy

Traditional Portfolio theory	Our use of Portfolio
Expected return on the nth asset E_N	Expected return on a strategy
Individual risk of nth asset R_N	Individual risk of a strategy
Expected return of portfolio R_P	Expected return of an architecture
Global risk of a portfolio RP	Global risk of an architecture

1. Expected return on the n^{th} asset = E_N

   In our use of portfolio, the E_N analogy corresponds to the "potential benefit" in architecting for and implementing a specific strategy towards the compliance requirement. More specifically, we view the architecture as a portfolio of strategies for satisfying requirements. We argue that the value of the architecture is in the value of the requirements it supports during the software system operation. In our use of portfolio thinking, the nature of the compliance and the case determines the dimensions on which the value of the potential benefit is to be realised.

2. Individual risk of nth asset = R_N

   In traditional applications, this variable is the historical price fluctuation of an asset. In our use of portfolio theory, the risk is formally modelled using the criticality and the likelihood of the risks occurring. The criticality of a risk indicates how bad the consequence of the risk is likely to be. Likelihood denotes the probability that the risk will occur.

3. Expected return of portfolio = $\boldsymbol{E_p}$

   The expected return of a portfolio is a weighted average of the returns of the individual strategies, tempered by their prioritised weights.

4. Expected risk of a portfolio = R_P

   The expected risk of a portfolio is a weighted average of the risks of the individual strategies, tempered by their correlations or covariance.

5. Expected Portfolio Cost

   Expected portfolio cost is the weighted average of the individual expected costs for implementing the different requirements and strategies.

3.3.6 Interpretation

The modification to the general portfolio optimisation formulation explicitly integrates aspects of portfolio valuation with a focus on evaluating compliance requirements in architectures. In the evaluation, costs and risks are associated with architectural strategies for satisfying compliance requirements. To capitalize on the architectural portfolio, requirements and architectural strategies have to be viewed as the pieces of a puzzle for creating value for the stakeholders. By having this broader view, it also becomes easier to find strategies that can be combined in order to reduce the risk since the entire architecture will be taken into account.

For an architecture, an optimal portfolio is described as one that satisfies all the requirements and constraints, and as such optimality is only assured if all possible requirements were considered in the generation of the portfolio.

The above analysis has shown that a portfolio approach can reduce risk and optimise decision-making for specific requirements in software systems. As long as the architectural strategy has a correlation below one with the expected return of the architecture, the risk will be reduced. The greater risk reduction is gained when the architectural strategy has a negative correlation with the expected return of the architecture.

3.3.7 Sensitivity Analysis

The portfolio analysis can be considered a multi-criteria decision problem. The purpose of sensitivity analysis is to determine how sensitive the analysis is to the uncertainty about key variables and assumptions in the input parameters. The objective is to provide an understanding of how the model responds to changes in input parameters. For example, the estimated parameters may be subject to uncertainty and parameters values could have been overestimated or underestimated. Sensitivity analysis allows for exploring the estimates and assumptions to understand how they affect the selections. There are considerable uncertainties in quantifying costs and even more uncertainty in completely understanding the risks. The optimal weights depend on the estimation of expected returns and risks. If these estimates are incorrect, the risk-return outcome may be very different from that envisioned. Put another way a small change in expected returns can radically alter the portfolio model. The model will benefit from explicit sensitivity analysis because of the possible inaccuracy of the utility function and changing priorities. We support the model with sensitivity analysis to increase the confidence in the model predictions and to provide a basis for "what-if" analyses.

We identify three main kinds of sensitivities in this thesis:

1. The sensitivity of the portfolio's proportions to the risks
2. The sensitivity of the portfolio's proportions to the correlations between architectural strategies
3. The sensitivity of the portfolio's proportions to expected return

Beta provides a measure of a financial portfolio sensitivity to change in the estimated value of the underlying parameters. Beta (β) measures the sensitivity to change or responsiveness in the return of an individual architectural strategy to the change in return of the architectural portfolio. In other words, the beta is a measure of the contribution of a strategy to the risk of the architecture portfolio.

A common expression for beta is

$$\beta = \frac{Cov\ (r_a, r_b)}{Var\ (r_b)} \tag{15}$$

Where Cov and Var are the covariance and variance operators. In portfolio, beta can be expressed as

$$\beta = \frac{Cov\ (x, y)}{Var\ (y)} \tag{16}$$

Where x and y, are unique architectures.

3.4 Related Work

This section provides an overview of closely related research on the use of portfolio theory in software engineering. It does not attempt to review portfolio theory and its general application as it is beyond the scope of this thesis. Portfolio theory is very well treated in [37] [55]. The use of portfolio thinking takes two main form: (i) quantifying investments in software in relation to market and (ii) understanding the nature of diversifying with the objective of optimising software portfolio decisions.

The link between selection of requirements and market value using portfolio has been explored by [166]. They proposed market driven, systematic, and more objective approach to supplement the selection of requirements, which accounts for uncertainty and incomplete knowledge in the real world-using portfolio reasoning. Sivzattian et al argued that argued that portfolio-based reasoning is well suited to inform the objective selection of requirements as it makes the connection between the selection decision and the market explicit.

Numminen [144] developed a model based on portfolio theory, which looked at strategies for software development that reduced market uncertainty. The model also explained why certain types of software functionalities are better combined with other types of software functionalities into software platforms in order to reduce market uncertainty. Numminen [144] presented a quantitative analysis that showed why and how a software platform strategy can reduce the uncertainty in the software development without an equal reduction in the expected return of the investment in the software development.

It is worth noting that the use of economic models to assess the cost and value of software requirements have been explored [104]. Karlsson and Ryan [104] used a cost-value approach for prioritizing requirements. They defined requirements value as the ability of a requirement to contribute to the customer's satisfaction with the overall system, when successfully implemented. A requirement's cost is an estimate of the additional cost required to meet that requirements alone. By relating requirements value to its cost, stakeholders have a measure of that requirement's ability to contribute to customer satisfaction. Analytical Hierarchy Process (AHP) [153] is used to calculate each candidate requirement's relative value and cost of implementation. These are then plotted on a cost-value diagram that serves as a conceptual map for analyses, discussion, and prioritization.

In contrast to these approaches, the value in the method presented in this thesis is in the architectural potential for satisfying the compliance requirement. The cost corresponds to the cost of implementing the architectural strategies to meet the specific compliance requirement. Karlsson and Ryan [104] acknowledged that the assessment of value and cost of implementation based on decision makers' "experience and judgment that this could be supplemented by other methods".

3.5 Summary

We have pursued a value-driven approach to address the requirements. We motivated the use of portfolio theory and have devised a model as a solution. We have described the approach taken, which is based on a simple and intuitive analogy modern portfolio theory. We reported on the model's formulation, its possible interpretation, and its sensitivity. We provided an overview of closely related work on the use of portfolio theory in software engineering.

CHAPTER 4

CONCEPTUALISING RISK WITH A COMPLIANCE DEBT METRIC

”Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted”

Albert Einstein

In the previous chapter, we pursued an economics-driven model that exploits portfolio theory to provide insights into allocating available resources in a manner, which optimises for costs, risk and value with respect to compliance. In this chapter, we pursue an economics-driven model that exploits the concept of technical debt as a metric for visualising trade-offs between short-term and long-term value in compliance.

First, we provide background on technical debt that is necessary to understand our model. We then describe the concept of compliance debt and its application leading to the model. We show how we have derived the model, and report on the assumptions that the model makes. Finally, we provide an overview of closely related work on the use of technical debt in software engineering.

4.1 Technical Debt: A Brief Background

4.1.1 Definition

The debt metaphor was first introduced by Ward Cunningham in 1992 [64] as follows:

”Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.”

Brown et al. [147] opined that “like financial debt, technical debt incurs interest payments in the form of increased future costs owing to earlier quick and dirty design and implementation choices”. The term has been broadly developed and it covers wider aspects associated with the overall systems development life cycle. Technical Debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of the system. The concept of technical debt has been defined from different perspectives based on context, domain and use. These are some of the definitions for technical debt used in the literature.

1. “The idea is that developers sometimes accept compromises in a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline), and that such compromises incur a ‘debt’: on which ‘interest’ has to be paid and which the ‘principal’ should be repaid at some point for the long-term health of the project.” [49]

2. “Technical debt is a term that has been used to describe the increased cost of changing or maintaining a system due to expedient short-cuts taken during its development.” [114]

3. “This includes things like bugs, design issues, and other code-quality problems that are potentially introduced with every addition or change to the code. These issues have a detrimental impact on developer productivity.”

4. “Technical debt describes the effect of immature software artefacts on software maintenance - the potential of extra effort required in future as if paying interest for the

incurred debt.” [160]

5. “The technical debt (TD) concept describes a trade-off between short-term and long-term goals in software development.” [190]

6. “The technical debt metaphor conceptualizes the trade-off between short-term and long-term value: taking shortcuts to optimize the delivery of features in the short term incurs debt, analogous to financial debt, that must be paid off later to optimize long-term success.”[142]

7. “The short iteration approach and the pressure to meet deadlines can create problems for the development. This pressure can encourage shortcuts concerning code maintenance that lead to accumulation of technical debt, that is, a backlog of deferred technical problems.” [176]

8. “Technical debt within the enterprise should be viewed as a tool similar to financial leverage, allowing the organization to incur debt to pursue options that it couldn’t otherwise afford.” [114]

9. “Making a decision about whether to fix or defer fixing a defect is important to software projects. Deferring defects accumulates a technical debt that burdens the software team and customer with a less than optimal solution.” [167]

10. “This may lead to increasing maintenance costs and the quality of the end product is undetermined. Furthermore, the cost increases the longer the technological shortcomings go uncorrected, until the code becomes unmanageable and essentially has to be completely rewritten. This escalating problem is often called design debt” [94]

11. “Architectural technical debt is incurred by design decisions that consciously or unconsciously compromise system-wide quality attributes, particularly maintainability and evolvability.” [125]

12. “Technical debt represents quality problems in software, and we use the quantification of such problems to determine the value of software.”[69]

13. “Technical debt is the sum of remediation costs for all non-compliances” [124]

Although these definitions look at technical debt from different perspectives based on context and use, a common theme is that technical debt is an indicator or a metric used to characterise technical compromises that can yield short-term benefit but may hurt the long-term health of the system.

4.1.2 What Problem does Technical Debt Address?

Technical debt addresses the problem of visualising and conceptualising the gaps in decisions between the current state of a software system and some hypothesized “ideal” state in which the system is optimally successful in a particular environment [49]. This gap includes items that are typically tracked in a software project, such as known defects and unimplemented features. There are often disagreements about important decisions regarding how to invest scarce resources in development projects, especially in relation to internal quality aspects that are crucial to the system [49]. These internal quality aspects such as documentation and code decay, are largely invisible to management and customers, and they do not generate short-term revenue [49].

Managing resource allocation to competing needs in system development has been acknowledged as a challenging problem [152]. This requires a means of assessing opportunities and gaps in creating value in software in terms of the cost and effort. Technical debt is used as a metaphor within software development to visualise and quantify the gaps in these decisions. Like financial debt, technical debt can be necessary in certain situations. Technical debt can be used as part of valuation tools to aid economic-driven software engineering. The technical debt metaphor can be used to aid decision-making in the selection and prioritisation process while linking these selection decisions to value-added.

4.1.3 Types and Applications

Technical debt has been applied to various software engineering problems. There are different ways of classifying technical debt. Steve McConnell [135] organized the debt into two groups: intentional and unintentional. Technical Debt can be unintentional due to bad engineering practices, and intentional when decisions are made to take on the technical debt for generating future value [135]. Intentionally incurred debt is based on a deliberate decision to optimize development for the present but to the detriment of another aspect at the long term. McConnell [135] makes an explicit difference between short-term, taken reactively for tactical reasons and long-term debt incurred proactively for strategic considerations. Additionally, Martin Fowler [82] classified the technical debts into: Reckless/Prudent and Deliberate/Inadvertent. These classifications make up Technical Debt Quadrant shown in 4.1. The technical debt quadrant can be used to classify the debt based on its characteristics e.g. if it was incurred intentionally. These classifications separate issues arising from recklessness from issues associated debt that was incurred strategically.

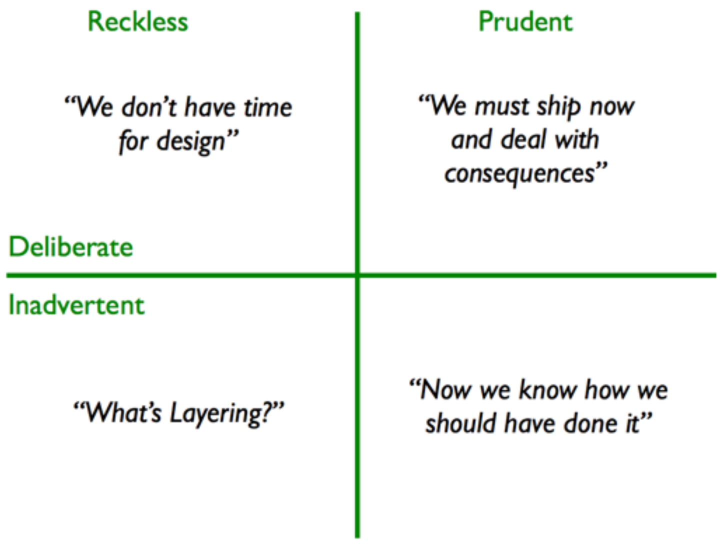

Reckless
Prudent
"We don't have time for design"
"We must ship now and deal with consequences"
Deliberate
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

Figure 4.1: Technical Debt Quadrant [82]

The following types of technical debt as well as their definitions were identified from literature:

1. Architecture Debt refers to architecture issues which can affect architectural requirements such as reliability, performance etc. [116] [49] [175]

2. Build Debt refers to build issues that make the build task harder, and consumes more time or processing unnecessarily [138]. A build debt may occur when a process runs an ill-defined dependency and becomes unnecessarily slow [4]

3. Code Debt is incurred when issues found within the source code, negatively affect the maintainability of the code. This is usually due to bad coding practises. [189] [36] [175]

4. Defect Debt is incurred when defects identified in a software project are not fixed due to competing priorities and limited resources [167]

5. Design Debt refers to technical debt that is incurred by violating the principles of good design practises [99] [189] Zazworka [175].

6. Documentation Debt refers to inadequate or incomplete software documentation. [175] [4]

7. Infrastructure Debt refers to an infrastructure issue that can delay or prevent any software development activity.

8. People Debt is incurred when people issues delay or prevent software development activities [175] [4]

9. Process Debt is incurred due to inefficient processes [60].

10. Requirement Debt is incurred when trade-offs are made with respect to what requirements to implement and how to implement them [116] [162].

11. Service Debt is incurred during the selection, composition, and operation of a web service [6]

12. Test Automation Debt is defined as the work needed in automating tests to support continuous improvement and integration [183]

13. Test Debt is incurred when issues with testing activities affect the quality of testing [175].

4.1.4 Properties of Technical Debt

To conceptualise and understanding technical debt, the following properties of technical debt were identified from literature.

1. Technical debt is relative to a given environment or domain [49]

2. Technical debt should be visible, so it can be considered in the decision making process [49]

3. Debt can be managed to create value. The value is the economic difference between the system as it is and the system in an ideal state for the assumed environment. Valuing technical debt for value is difficult. [49]

4. Debt does not necessarily combine additively, but taking on too much debt is detrimental to a system. [49]

5. It is important to differentiate between strategic debt and unintentional debt. [49]

6. Technical debt has an associated cost. Technical debt is said to have a negative impact on productivity, quality and risk [175]. There is a challenge in quantifying technical debt.

7. Interest is the extra effort required to pay back the technical debt by modifying the system [126]

8. Principal is the estimated cost of resolving the technical debt [126]

9. Debt amnesty – where accrued technical debt can be thought of as written off and does not have to be repaid. Incurring technical debt does not always result in an obligation to be repaid; rather it might create opportunities to invest without the obligations. [49] [175]

4.1.5 Valuations

Technical debt can be valued using a variety of techniques. These techniques make different assumptions and require different tools to capture uncertainty. These approaches aim to quantify the benefit and costs associated with technical debt in a system. The approaches used are mainly in the following categories as identified by [126]

1. Calculating technical debt using mathematical models: For intentional technical debt, the decision-making process about which debt to incur is fundamental, because this decision is about balancing cost and value [116]. Some of the approaches exploiting financial models include Simple Cost-Benefit Analysis [90] [161] [157], Analytic Hierarchy Process, Portfolio theory [159] [161] [25] and Options [161] [5] 2013a; Seaman et al. 2012).

2. Calculating technical debt using metrics of source code [36] [97] [80]

3. Estimate various types of the cost of handling the incurred technical debt [167]

4. Human estimation according to expertise and experience [160]

5. Calculating the distance between the actual solution and the optimal solution [73] [25]

6. Estimating technical debt using quality metrics: In the model of CAST [155] coding violations associated with the quality attributes are used to calculate TD. The TD model of [15] supports the calculation of both, remediation and non-remediation costs of debt incurred by technical quality issues. In [123] [124] the authors presented SQALE , a quality model that also provides a technical debt calculation model.

SQALE calculates the total of the effort required for fixing all violations of quality requirements.

4.2 Compliance Requirements: A Compliance Debt Perspective

Most of the research works reviewed linked technical debt to code writing and code development. Unlike previous works, we introduce a new form of architectural debt linked to compliance requirements. Either a compliant system can be interpreted as a system being compliant to some specific compliance requirements, or the system's purpose is to aid compliance to compliance requirements. We extend the technical debt definition and define compliance debt as the gap between what can be achieved with the available resources and the hypothesised "ideal" environments where the compliance requirements are successfully and completely satisfied. Our definition differs from previous definitions as we view debt as any decision that prevents compliance requirements from being completely realised. Defining that debt as a gap simplifies quantifying and making the debt visible. We introduce technical debt as a decision metric in evaluating

We incorporate the debt analysis at the requirement analysis and early architectural decision levels. Technical debt can inform the architectural evaluation process and the decision for investing in a specific architecture at early stages of the development lifecycle. Our objective is to avoid inappropriate selection of an architecture that is not value driven and debt aware. The key principle here is to tackle and manage the increased and unjustified debt, which can be associated with the selection and consequently the inappropriate architecture candidates. We assume that debt can vary with different architectures and each architecture can deliver its own trade-offs for risk, value, cost and debt reduction. We advocate a predictive approach for anticipating and managing debt at the requirements and early architectural evaluation stages. A predictive approach can be applied during the early stages of the engineering process to predict the debt, its impact, when it will be incurred, when it will be paid off, and the interest if any. Classical approaches to managing technical debt in software development lifecycle tend to be retrospective. Unlike retrospective approaches, predictive approaches allow planning.

Technical debt in compliance requirements can be traced back to requirements – the way requirements are engineered, elicited, selected, prioritised and analysed. Debt can be linked to the alternative architectural strategies used to operationalised requirements, their appropriateness, resources used, and risk involved. The interest on this debt can be characterized as the rate of increase in this distance. Uncertainty about whether or not a decision is appropriate will have an associated penalty, which may incur a debt. In this sense, technical debt can be considered as a particular type of risk; the problem of managing compliance debt boils down to managing risk and making informed decisions [20].

We define compliance debt as:

The gap between the "ideal-level" and the "actual-level" of compliance of the selected architecture

In relation to the architectural portfolio, we have an optional portfolio and the portfolio selected, for short-term gain. We view investments in individual requirements and architectures as a loan, which may incur interest with time. Interest may be incurred on a compliance debt, if the debt is not properly managed by unlocking the potential for value creations. To achieve a satisfied level of compliance, we improve the utility on a specific compliance dimensions, as dimensions impact the utility of the architecture.

Compliance debt like technical debt can be unintentional due to bad engineering practises or intentional. Intentional compliance debt occurs when we decide to take on the debt as a strategic tactic for gaining value. Compliance debt can be incurred in the following situations:

1. Budget restrictions: sometimes, when architects need to select a specific portfolio, budget limitation can restrict the selection process. Budget restriction has different extremes that lead to technical debt in compliance. Budget restriction may influence

the architect's selection decision by selecting an unsuitable portfolio that does not satisfy the compliance requirements, but is within the budget. Secondly, debt can be incurred when the selected portfolio is expensive and its cost outweigh the portfolio's expected returns with respect to compliance.

2. Over-architecting and Underutilisation: Debt can be related to situations where the portfolio is over architected to satisfy compliance requirements. This may lead to underutilisation of the system to be developed from the architecture.

3. Under architecting: Debt can also be related to situations where the architectural portfolio does not match the compliance requirements because of inappropriate requirement engineering.

4. Schedule constraints: Poor and quick decisions may add value in the short-term, but can introduce long-term debt in situations where the software is non-compliance and replacing the software is inevitable. Sometimes the stakeholders may avoid proper compliance requirement engineering and architectural evaluation due to time restrictions.

5. Compliance debt during design-time can be incurred debt due to lack if long-term vision and planning.

6. Requirements mismatch can lead to debt when the selected web architectural portfolio does not fully match the compliance requirements for the system.

7. Debt can be incurred due to the difference in "expected-level" of utility and "actual-level" of utility

As mentioned in previous chapters, approaches to managing compliance to compliance requirements can be predictive or retrospective. We contribute to the predictive approach by using value-based reasoning to aid prediction and decision-making. We examine the likely risks and benefits associated with each requirements and value the extent to which a portfolio of requirements means the overall goal of compliance.

4.2.1 Why a Debt Perspective?

The technical debt analogy is well suited to address the problem of visualising trade-offs from a value-based engineering perspective. To understand the value of an architecture in respect to specific compliance requirements using an economic approach, we need a valuation metric that is suitable for strategic valuation that accounts for uncertainty and makes the value of a trade-off tangible. Using a compliance debt metric satisfies these requirements.

The debt analogy provides an analysis paradigm that conceptualises the value of compromises that can yield short-term benefit but may hurt the long-term health of the system [126]. It provides a way to visualise the gaps in decisions between the selected architecture and some hypothesized "ideal" state in which the architecture completely satisfies the relevant compliance requirements. In traditional applications, intentional technical debt can be used to recognise decisions that optimize development for the present to generate value, which may be to the detriment of another aspect at the long term [134] [135]. Compliance debt is a form of strategic debt that is descried as "long-term debt, usually incurred proactively, for strategic reasons" [135]. We look at compliance debt, as a strategic debt that it is accrued for the purposes of maximising value. This debt has been quantified, and is more visible.

Prioritisation is a precedent for compliance debt. It is important to prioritise the compliance risk associated with an architectural portfolio. This can be done using a compliance debt metric. When it becomes necessary for stakeholders to prioritise for value, to deliver required amounts of functionality within specified constraints such as cost, teams will need to make trade-offs. These trade-offs often create compliance debt. Pragmatism and prioritisation are strongly linked as one prioritises when being pragmatic, while effective prioritisation involves a level of pragmatism [175]. It can be easily seen that pragmatism can also lead to the selection of sub-optimal architectural portfolios to meet budgetary constraints.

4.2.2 The Model

Like technical debt, the measurement of compliance debt is difficult. Following on from our portfolio model developed in chapter 3, identifying the optimum architecture is a core activity in the evaluation process. An architecture is optimal for a given problem if it achieves minimum risk and maximises value while satisfying critical constraints. This is defined as a portfolio optimisation problem to minimise risk and maximise return subject to cost constraints. Unlike quantifying technical debt with line of code, quantifying compliance debt is difficult. The Sharpe ratio is a measure of the utility of the architecture with respect to specifed compliance requirements [163]. Where E_R is the expected return and P_d is the standard deviation of the portfolio, we define this in terms of Sharpe ratio.

$$Maximise\ \mathrm{E_p}/\mathrm{P_d} \tag{1}$$

Subject to lower limit $\leq$ *Cost* $\leq$ *upper limit*

This can also be used for comparison and benchmarking of other architecture portfolios. The architecture with a high Sharpe ratio is the one that achieves the best combination of value and risk in presence of uncertainty.

We quantify compliance debt in order to inform architectural portfolio selection decisions. Once the selection decision is made, the technical debt is said to be active and needs to be managed. We look at compliance debt in order to anticipate the value of a candidate architectural portfolio by quantifying the cost of achieving the intended utility-level. The aim is to quantify the debt on the architecture during the selection process. We look at compliance debt from the satisfactory utility-level perspective. We define and deal with two different values for compliance debt during design-time: (i) Debt value, (ii) Debt principal.

In order to accept a specific amount of compliance debt, we define debt threshold and satisfaction level of portfolio's value. This is expressed as the Sharpe ratio and it is a measure of impact on the utility of the architecture. Here, the difference between the utility "expected-level" and "actual-level" is triggering a debt. After quantifying compliance debt, we use this quantified value as a metric. Subsequently, this metric can be used by decision makers against other trade-offs in order to quantify the decision of selecting a specific architectural portfolio. We rank the architecture by considering the amount of debt that each candidate solution may incur as the deciding factor. From our earlier explanation of the debt using the technical debt metaphor as the gap between what can be achieved with the available resources and the hypothesised "ideal" environments where the goals are "optimally" achieved, we formulate the value of the debt principal:

$$T_D = I_{RT} - R_T \tag{2}$$

Where T_D is the debt principal, I_{RT} is the Sharpe ratio of an "optimal" architecture and R_T is the Sharpe ratio of the selected architecture. The ideal value is depends on the application and business context. T_D for any other tactic is calculated as the gap between value of architecture k (I_{RT}) and the value of the architecture in question. The architectures can then be ranked from best (low T_D) to worst (high T_D). This model indicates that the analysis and measurement of TD-Principal and it aims to guide critical management decisions about how to allocate resources to reduce compliance risk and optimise for value.

This approach models compliance debt as a function of compliance requirements trade-offs in the architecture. It recommends the desirable cost-effective solutions for given time of interest for specific requirements. This can incur technical debt in the implementation and though we may implicitly know that a particular architecture is not optimal in the long-term, presumably the choice of the architecture is justified as it is fit for purpose and relative to the scenarios of interest. It is only when time passes that debt begins to accrue as the compliance and business needs begin to favour the "optimal" solution more than the actual solution. With respect to "interest" on the debt, which we loosely translate as the cost of neglecting the debt, this can reflect on the rate of change between the current solution and the improved one along with the related ramifications.

4.2.3 Valuation Issues and Assumptions

1. Another assumption used in this analysis is that the market value of the debt is equal to the expected value of all future interest and principal payments. This implies that the debt is risk-neutral and expected returns on the architecture is equal, regardless of the time profile of the system.

2. We assume that the higher the level of portfolio returns with respect to compliance, the smaller the compliance debt. This assumption was justified by previous research, which reveals that performing changes on systems with higher code quality is more efficient [143]

4.3 Related Work

Brown et al.[49] recognised the need for models and techniques, which can identify and manage technical debt, its causes, and its trade-offs based on the economic impacts. They argued that compromises should be taken in a way where there should be a balance between the short-term deadlines and the long-term sustainability. Managing technical debt is crucial as technical debt may cause long-term problems if unmanaged, such as increasing the maintenance cost. They have associated the relation between the technical debt and the financial debt as they considered that it to be the increase of future maintenance cost, due to earlier quick and dirty design and implementation choices.

Different approaches have been used for managing technical debt at different phases such as identification, measurement, prioritisation, prevention, monitoring, repayment, representation and communication. Approaches used to identify technical debt include analysing the code to identify code violations [22] [53] [65] [66] [94] [99] [124] and analysing dependencies between different types of software elements [48] [124] [190] [136]

Technical debt prioritisation approaches include cost benefit analysis [167] [188], high remediation cost first [124], portfolio approach [159] and high-interest rate first [88].

Approaches used for measuring technical debt in the literature include using calculation models [62] [66] [69] [94] [124] [136] [142] [143], code metrics [22] [80] [80], human estimation [160] [159], cost categorisation [167] and solution comparison by calculating the distance between the actual solution and the optimal solution [73].

Brown et al. [49] proposed an approach for managing technical debt on the architectural level. The approach was built on the idea of technical debt assessment in an iterative release planning process based on dependency analysis. They introduced a dependency mapping method for analysing three types of dependencies in the iterative release planning, which are the dependencies between requirements using DSM, the dependencies between requirements and architectural elements using DMM, as well as the dependencies between architectural elements using DSM. They investigated two paths in the view of value-maximization and cost minimization. According to their objectives, the path with the highest value and least architectural elements dependencies is the first one to be developed. Their method aims to help in decreasing the accumulating debt and can contribute to architectural decision-making. Based on [49], Nord et al. [142] looked at a new metric for managing the structural technical debt based on the propagation cost, which is called Change Propagation Metric. Schmid [158] proposed a method for formalizing technical debt using a cost perspective with formal analysis.

However, none of the available approaches in the literature has mentioned technical debt in portfolio selection and their link to strategic value-added. Furthermore, there is a general lack for models and techniques for identifying and quantifying debt at the architectural level. The work presented in this thesis is an early attempt at characterising compliance debt and understanding how the debt can be quantified in architectures.

4.4 Critical Evaluation

The model presented in this chapter exemplifies the need for a framework, which measures compliance debt and reveals the rationale behind it. Considering debt in engineering for compliance is motivated by the need to take strategic tactics to satisfy compliance requirements for maximising value. In this sense, we introduced a new type of debt, which is at the architectural level for visualising gaps in compliance. This type of debt is a strategic debt that can be attributed to the gaps between an ideal architecture satisfying all compliance requirements completely and the selected architectural portfolio. The debt can be attributed to different situation such as poor architecting, budget restrictions, not satisfying the requirements and poor decision-making. Technical debt concept is a metaphor that has been examined in different fields taking into account different dimensions. However, none of the available work has discussed technical debt with respect to specific requirement concerns such as compliance at the architectural level. We argue that compliance debt can be used as a metric to aid decision-making with respect to maximising value. Our work is different from the available approaches in some point as follows:

1. First, from a concept point of view, when taking on technical debt on the current architecture, we assume that the cost today will be constant until the system is retired.
2. Second, from a compliance debt perspective, we are managing debt by taking decisions on architectural-levels. We are dealing with the selection decision as a long-term investment, which considers technical debt strategically and proactively.

4.5 Summary

We have pursued a debt-aware approach to address the requirements in the form of a compliance debt metric. We have motivated the use of the technical debt metaphor for evaluating an architecture for compliance and have devised a model, as a solution. We have described the approach taken, which is based the technical debt metaphor. We have reported the technical debt definitions and types of technical debt in the literature. We have introduced and defined a new form of architectural technical debt, linked to compliance. Finally, we have provided an overview of closely related work on the use of different technical debt approaches in software engineering.

CHAPTER 5

A SYSTEMATIC METHOD FOR EVALUATING COMPLIANCE IN SOFTWARE SYSTEMS

The whole of science is nothing more than a refinement of everyday thinking."

Albert Einstein

In the previous chapters, we have presented the portfolio-based model and a debt model for maximising value and reducing risk when managing compliance requirements. In this chapter, we support the model with a five-phase value-driven method. A software engineering method is "an approach to perform a system development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products" [47].

This chapter describes a method for evaluating compliance that combines techniques that have been developed throughout the course of this research. The main aim of this research is to provide a framework that gives requirement engineers with a set of guidelines to inform the decision of investment in compliance, derive more realistic compliance requirements based on their economics, risks and technical debt for any given project. In order to do this, our approach consists of applying value-driven techniques to improve the requirement decision-making process. This chapter describes a method that combine the concepts and models described in the previous chapters.

5.1 Overview of the method

The method defines a set of steps that describes possible ways for estimating the model parameters, applying the models and performing the analysis. An overview of the method is illustrated by Figure 5.1. The method is a systematic and iterative method structured into five phases. Each phase of the method presents a number of techniques and guidelines to assist in the analysis of compliance requirements. Figures 5.2, 5.3,5.4,5.5 and 5.6,depict an overview of the method phases.

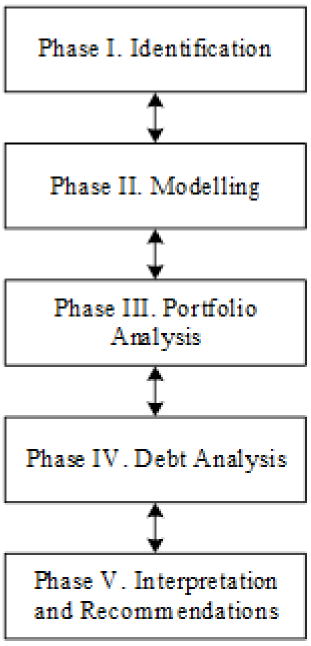

Phase I. Identification
Phase II. Modelling
Phase III. Portfolio Analysis
Phase IV. Debt Analysis
Phase V. Interpretation and Recommendations

Figure 5.1: Overview of the method

The **first phase** is the identification of goals. This phase starts with setting the objectives for the evaluation and the desired compliance threshold. During this phase, priorities are assigned to goals and interactions between goals are identified. The method relies on existing goal-oriented requirements engineering method [7] to elicit and analyse compliance goals.

Goals can be refined into sub-goals and domain assumptions. Domain assumptions are statements about the application domain that are outside the control of the system [170]. A goal could be further refined to correspond to one or more further sub-goals that may need to be realized or could affect the software system.

Phase I. Identification of Goals

Input: High-level goals and objectives

Process:

- Specify compliance goals
- Set the objectives for evaluating and desired regulatory compliance threshold
- Trace goals to architecture

Output:

An elaborated goal model

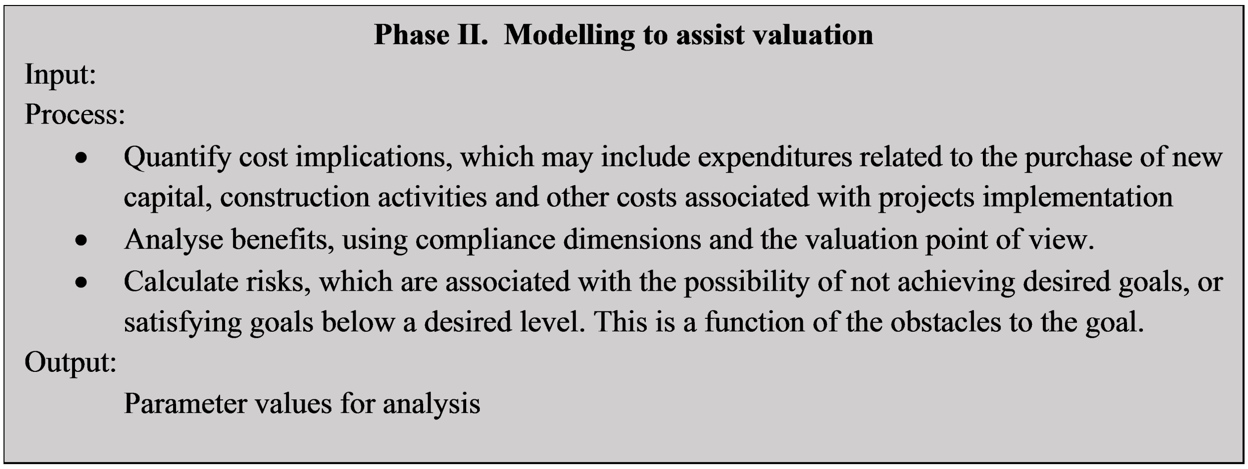
Figure 5.2: Phase I - Identification of Goals

In the **second phase**, we present the modelling to assist with the valuation of the model's parameters. Our method is however not constrained to the use of any particular valuation. We use a multi-perspective valuation points of view framework for valuing the impact architectural strategies on the compliance requirements. We analyse the associated costs, benefits and risks. The valuation used requires a comprehensive solution that incorporates multiple valuation techniques from different value perspective and impact parameters, some with subjective estimates, and others based on historical data, when available.

It is important to know whether the expected overall benefit resulting from a decision outweighs the expected overall cost.Risks are associated with the possibility of the architectural strategy not satisfying the desired compliance goal or satisfying goal below a desired level. In this method, risks are modelled as a function of obstacles.

In the **third phase**, we assess for desirability by evaluating different portfolios. We use portfolio theory to identify an optimal portfolio of strategies (requirements) with a total value that minimises risks and maximises return. We shortlist valid and optimal architectures. A portfolio is valid, if it satisfies the relevant constraints such as cost, risk, schedule or compliance requirement threshold. The ideal approach is to short list valid portfolios that maximise expected net benefits and minimise risks on dimensions related to the compliance requirements.

In the **fourth phase**, we evaluated the debt associated with each portfolio. The ability


Phase II. Modelling to assist valuation

Input:

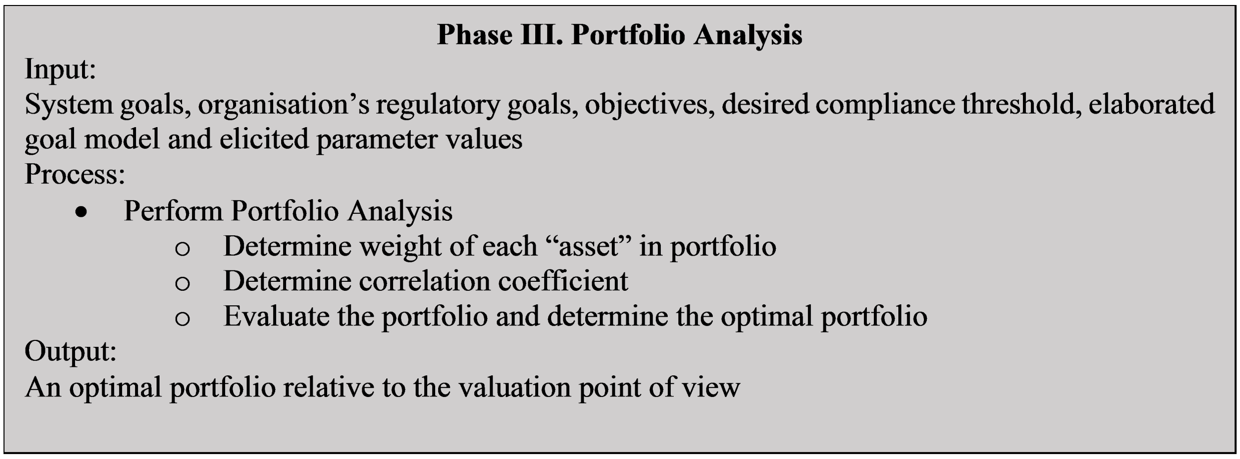
Process:

- Quantify cost implications, which may include expenditures related to the purchase of new capital, construction activities and other costs associated with projects implementation
- Analyse benefits, using compliance dimensions and the valuation point of view.
- Calculate risks, which are associated with the possibility of not achieving desired goals, or satisfying goals below a desired level. This is a function of the obstacles to the goal.

Output:

Parameter values for analysis

Figure 5.3: Phase II - Modelling to Assist the Valuation


Phase III. Portfolio Analysis

Input:

System goals, organisation's regulatory goals, objectives, desired compliance threshold, elaborated goal model and elicited parameter values

Process:

- Perform Portfolio Analysis
  - Determine weight of each "asset" in portfolio
  - Determine correlation coefficient
  - Evaluate the portfolio and determine the optimal portfolio

Output:

An optimal portfolio relative to the valuation point of view

Figure 5.4: Phase III - Portfolio Analysis

to find the optimal solution is complemented with the ability to rank portfolios. We rank the portfolios by considering the amount of debt that each architecture may incur as the deciding factor. We calculate and assign the compliance debt of each alternative. From our earlier explanation of debt using the technical debt metaphor as the gap between what can be achieved with the available resources and the hypothesised "ideal" environments where the goals are successfully achieved. The ideal value is context dependent. The ideal value is application and business dependent. T_D for any other tactic is calculated as the gap between value of architecture k_{IRT} and the value of the architecture in question. The architectures can then be ranked from best (T_D) to worst (T_D).

Phase IV. Debt Analysis
Input:
An optimal portfolio relative to the valuation point of view
Process:
- Calculate the debt associated with each portfolio
- Rank portfolios

Output:
A ranking of the different portfolio according to debt

Figure 5.5: Phase IV - Debt Analysis

In **fifth phase**, we interpret the results and give recommendations relative to the set objectives.

Phase V. Interpretations and Recommendations
Input:
A ranking of the different portfolio according to debt
Output:
Interpret the results and give recommendations relative to the set objectives

Figure 5.6: Phase V - Interpretation and Recommendations

5.2 Main Principles

The method is based on the following principles:

1. **Systematic and well defined** - The method provides a systematic and well-defined approach to examine both technical and non-technical aspects that influence the decision-making process.

2. **Interactive Process** - The phases of the method follow an interactive modelling approach. For explanation purposes, each phase is described separately but in practice, processes are highly intertwined.

3. **Flexible** – The method can be adjustable to accommodate different project and regulatory compliance needs. Depending on the specific compliance requirement

and the amount of resources allocated, different techniques proposed in method can be applied

4. **Goal-oriented requirements engineering** - An important characteristic of the method is the adoption of goal-oriented approach. Goals provide a suitable mechanism to specify requirements.

5. **Qualitative and quantitative decision-making** – The method provides quantitative and qualitative techniques to inform the decision-making process. The ranking of the candidate portfolios is quantitatively obtained by using typical multi-criteria decision-making techniques.

6. **Intentionality:** Engineering and evaluating compliance requirements should be such that the system, once designed and implemented, will behave in the state described by the compliance document [164] [165] .

7. **Ability**: Any solution is acceptable only if the agent operationalising the compliance requirement has the capability to perform the necessary actions to satisfy that requirement [165].

8. **Traceability**: The ability to trace compliance decisions to the relevant guidelines is essential. Information on traceability is the first element of proof to support compliance choices.

The first phase of this method consists of identifying the decisions to be taken and eliciting the compliance goals to evaluate these decisions against.

5.3 Phase I: Identification

The first phase of this method consists of identifying the decisions to be taken together with their scenarios, defining the goals against which to evaluate these decisions.

5.3.1 Specify Regulatory Goals using Existing GORE approach

Specifying stakeholder requirements is considered the first activity of any system development [145]. In this step, we identify the required compliance requirements, which are critical to the evaluation and to the set objectives. A question of interest is how can we elicit and model these regulatory requirements?

In order to analyse these compliance requirements, we need to improve our knowledge about the system, its domain, and its system goals. The first step towards this objective is to build a goal model without compliance in mind. We use existing goal-oriented requirement engineering methods [117] to elicit, elaborate, and refine the goal model. The objective is to quantify the cost, risk and value of compliance to a specific compliance requirement, which we will explore in Phase III.

The method is more concerned about *how* the compliance *goals* in a given scenario are satisfied and this affects the architecture of the software system. The objectives are (i) to provide a paradigm, which traces the compliance requirement, written in the regulatory text, to its architectural elements. Though existing architectural evaluation make use of scenarios, they lack the support for systematically analysing and tracing requirements to architectural strategies for the architecture. In existing architectural evaluation methods, the architect explains how relevant architectural decisions contribute to realizing a particular requirement.

This method builds on well-established practise from Goal-Oriented approaches to Requirements Engineering (GORE) [117] to specify the requirements. The use of goal-oriented requirements modelling allows the evaluation team to specify stakeholder needs at different levels of abstraction. Goals can range from high level, strategic objectives (such as "Comply with the data protection act") to low level, operational concerns (such as "comply with the data protection by supporting authentication"). An important observation is that higher-level goals are more stable than low-level ones, and hence are less likely to change over time. The decision to specifying generic or restrictive goals depends on the stage of the evaluation. Generally, in the beginning of the process generic goals are more suitable, while at later stages, more discriminative goals should be preferred.

To decide if a system is compliant, we need to understand

- The functional and quality goals of the system affected by the compliance requirements
- The characteristics of the application domain
- The system's design
- The acceptable level of compliance for the quality goals satisfaction considering the domain and design variations.

There are three steps involved in the specification phase, which should be performed in incrementally and iteratively.

Acquire high level goals The first step of the goal specification phase consists of eliciting key organizational objectives and constraints as well as understanding the application domain. To obtain high-level goals, typical elicitation techniques can be used.

Refine Goals Next, the high-level goals are refined. The objective of this activity is to represent goals in a more precise and measurable way. Goals are structured in a refinement tree such that high level goals are decomposed into more concrete sub-goals that correspond to richer and more realistic representations of parent goals. The goal refinement tree captures relationship among goals using AND/OR refinement. AND refinement relates to a goal that will be satisfied when all its sub-goals are satisfied. OR refinement refers to a goal that is sufficiently satisfied if at least one of its sub-goals is satisfied. The refinement of goals continues until it is possible to measure their satisfaction objectively, at this stage sub-goals are called operational goals. The more a goal hierarchy is decomposed and goals become more tangible, the easier it is to identify metrics to assess operational goals. However, the evaluation team needs to balance the benefits to have a comprehensive goal specification against the time and effort necessary to build the tree refinement.

Identify Interactions Between Goals As we get a better understanding about stakeholder's needs through the process of goal refinement, it is quite natural to find interactions and dependencies between goals. These interactions normally occur between quality goals and can be either positive or negative. Alves [3] defined some heuristics to guide the identification of interactions between goals:

1. Does the achievement of this goal depend on the completion of another goal?
2. Does the achievement of this goal increase the satisfaction of another goal?
3. Does the failure of this goal cause the failure of another goal?
4. What other goal must be achieved in order to satisfy this goal?

Negative interactions show conflicting situations between goals. A conflict exists between two goals when one goal interferes with the satisfaction of another goal[3] .

1. Does an increase of satisfaction of this goal cause a decrease of satisfaction of another goal?
2. Does a decrease of satisfaction of this goal cause an increase of satisfaction of another goal?
3. Does the satisfaction of this goal hurt the satisfaction of another goal?
4. Does the failure of this goal increase the satisfaction of another goal?
5. Does the interaction between goals prevent their mutual satisfaction?

By answering such questions, stakeholders can reason about priorities for the compliance goals.

5.3.2 Set the objectives for the Evaluation

For this step, the process entails identifying the objectives that the evaluation needs to address. In the previous chapter, we highlighted several uses of the portfolio model for addressing some representative compliance requirement problems. The objective for conducting the evaluation will be tailored to the said problem. Understanding what drives the evaluation is essential for:

1. Identifying the compliance requirements that are critical for analysing the said objective, which will be explored in this phase;
2. Understanding the context of the systems the compliance requirements will be implemented
3. Identifying both the value of the requirement relative to the valuation dimension(s) and impact criteria on which the requirements need to be assessed, which will be explored in phase III; and
4. Interpreting the valuation results relative to the said objectives, which will be explored in Phase V.

Below are the possible drivers for initiating the evaluation for compliance requirements for value

1. Valuing the cost-effectiveness of designing/implementing for compliance
2. Requirement and implementation risk assessment.
3. Trade-off analysis by comparing two or more portfolios and understanding the debt each portfolio incurs

In this step, we also define the satisfaction functions. Once goals are refined and appropriate metrics identified, we now examine how the satisfaction of such goals can be measured. In the context of the evaluation, satisfaction function is a measurement that expresses the relative value in which an architecture brings towards the satisfaction of goals.

5.3.3 Trace the Goals to the Architecture

A feature of GORE models is their built-in vertical traceability – from strategic business objectives to technical requirements to precise specifications to architectural design choices. The ability trace high-level compliance goals to the corresponding architectural elements is helpful, as one of the criteria for managing compliance is traceability.

An output from the previous step is the refined goals that need to be satisfied. In the refinement process, the goals are decomposed into more concrete sub-goals, which correspond to with tangible representation of the parent goals. In this thesis, the refinement is done using guidance on how it could be operationalised by the architecture in association with the solution domain. Another objective of the refinement process is to make compliance goals as measurable as possible to quantify the costs and benefits associated with goal.

The refinement process could result in: (i) identifying the architectural elements (strategies) responsible for operationalising the goals; or (ii) identifying architectural elements, which might be impacted by the goals. We then refine the goals using knowledge of the solution domain until a trace with the associated architectural artefacts, which are to be impacted by the goal, is established. We will not go into further details as that is outside the scope of this thesis.

5.4 Phase II: Modelling to Assist Valuation

Evaluating compliance in an architecture using a value-driven approach needs a comprehensive solution that is flexible enough to use different valuation techniques. Techniques that support subjective estimates or estimates based on historical data, when available. This is because of the following reasons:

1. First, the valuation activity is a human-centred activity that implies subjectivity

and introduces different perspectives to the valuation problem.

2. The valuation is relative to the evaluation objectives, which was set in Phase I and the primary drivers motivating the need for compliance. This necessitates a valuation solution that is flexible enough to capture the value relative to the said drivers.

As a compromise, the problem of valuing an architecture for compliance requires a comprehensive solution that is flexible enough to use different perspectives and to incorporate multiple valuation techniques. To address this problem, we outline a conceptual valuation points of view framework using perspectives from the software value map [87]. The framework aims to capture and value the impact of the compliance goal on the architecture from different points of views.

To analyse compliance goals, we employ parallels with CBAM [98] [141] in modelling and valuation. The Cost Benefit Analysis Method (CBAM) is an architecture centric method for analysing the costs, benefits, and schedule implications of architectural decisions. The CBAM consists of the following steps:

1. choosing scenarios and architectural strategies (AS);
2. assessing Quality Attribute(QA) benefits;
3. quantifying the Architectural Strategies;
4. assessing costs and schedule implications;
5. calculating desirability;
6. making decisions.

CBAM guides stakeholders to determine a set of architectural strategies that address their highest priority scenarios. The chosen strategies represent the optimal set of architectural investments based on considerations of: benefit, cost, schedule, within the constraints of the elicited uncertainty of these judgements and the willingness of the stakeholders to withstand the risk implied by uncertainty.

We use some of the steps of CBAM to assist in our modelling and valuation. The steps based on CBAM involved in this phase are:

1. Identify and quantify benefits
2. Quantify Cost Implications
3. Quantify Risk Implications / Risk Analysis

We use CBAM principles to track benefits on goals/scenarios along with the compliance goals. The benefit or the effectiveness of a goal or strategy is an assessment of how well the strategy helps in achieving the compliance goals.

5.4.1 Identifying and Quantify Benefits

The first step of CBAM consists of identifying the decisions to be taken together with their architectural strategies (AS), defining the goals against which to evaluate these decisions. In classical CBAM, quality attributes are assessed by assigning a utility value for the architecture decision response to relevant scenarios. We extend classical CBAM by assessing the utility of the scenario with respect to its impact on the different domain dimensions. We aim to identify architecture design decisions, which minimize costs, reduce risk and maximize value on some identified compliance dimensions of interest. Once the scenarios have been developed and their values elicited, their relationship to the different compliance domain dimensions is determined. The degree to which each of these attributes is expected varies from domain to domain and tends to benefit from an expert's input. The evaluation benefiting from these dimensions shall be long-term and strategic in nature. In this step, we describe a systematic process to quantify the benefits of the architectural strategy or value of the goal. The process consists of the following steps:

Obtain the Domain-Specific Dimensions to Determine Utility Once the compliance goals have been specified, its relationship to the domain specific dimensions is determined. For this framework, we use a simple approach inspired by the GQM method [23] for tracing the compliance goals to the domain specific dimensions that are intended to define the goal operationally. This helps interpret the goal with respect to predefined compliance dimension. This approach gives a more comprehensive picture of the goals and dimensions used for the valuation. The measurement scale is used to quantify the impact and utility an architectural strategy has on the compliance dimension, which in turn affects the overall impact on compliance. The approach used is made up of the goals, dimensions and measurement scale as shown in figure 5.7.

The degree to which each of these attributes is expected, varies from application to application and the specific domains. In the information security domain, the criteria used will include checking how much the goal being evaluated influences parameters such as confidentiality, integrity and availability. In the sustainability compliance domain, the criteria used will include checking how much the goal being evaluated is likely to impact and contribute to compliance dimensions such as human, environmental and economic sustainability

Usually qualitative metrics may be extended by quantifying or replacing value such as high, medium or low with numerical weights. Although this principle is often used in practice, problems such as lack of precision and lack of clear criteria to validate the estimation still exists, as this technique is subjective. For simplicity, we propose quantitative measurement metric that considers the impact of the goal on different attributes (e.g., compromise of integrity and confidentiality) based on subjective criteria such as the stakeholders' experience. The parameters value range from one to three is assigned depending on if the impact is high, medium or low. For the valuation to be accurate, it is important to estimate the parameters accurately. In this thesis, our focus is more on developing a framework for quantitative analysis system rather than model parameterization. However, evaluating the sensitivity of model to variations in model parameters will be valuable. In chapter 6, we evaluated the sensitivity of the model to various parameters.

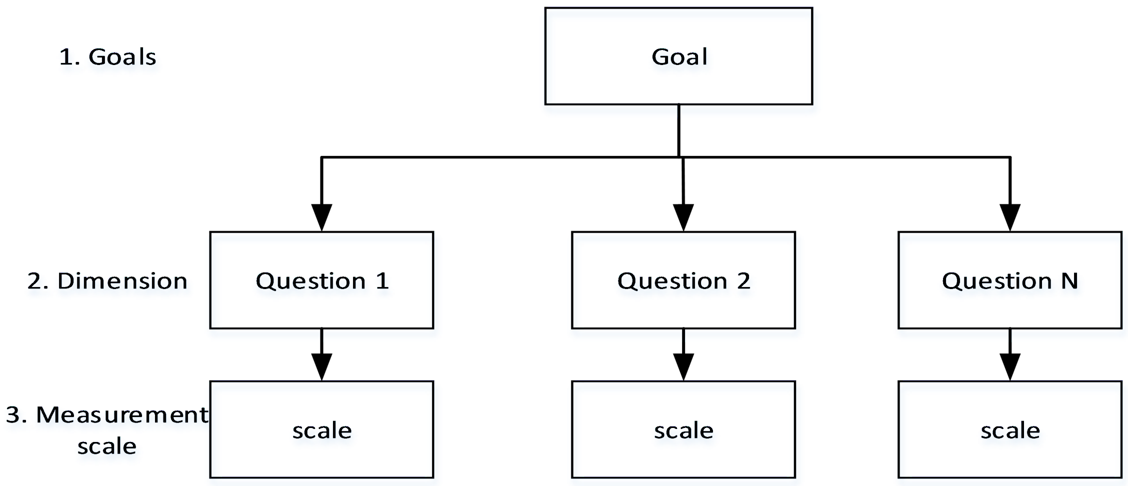


Figure 5.7: Tracing compliance to domain specific dimensions

Valuation Perspective to Determine Utility We use the Software Value Map [110] to identify the different value perspectives. These perspectives that can be used to evaluate the impact of an architecture strategy on these dimensions. This technique offers a unified view of value based on four value perspectives: the financial, the customer, the internal business process, and the innovation and learning [110]. For our work, we use three perspectives as it is difficult to quantify the value of innovation and learning.

1. The financial perspective looks at aspects of the implementation decision that affects short and long-term financial goals [110]. Some of the most common financial measures are cost of implementation, operation, maintenance and evolution, and/or profit margins among the others.

2. The customer perspective looks at the value proposition of the software to customers [110]. Customer perspective on sustainability can measure the value that an architecture strategy can deliver to the customer with respect to the perceived time, quality, performance and service, and cost. ?

3. The internal business process perspective is concerned with the processes that maintain and encourage sustainability. Quality, cycle time, productivity and cost are some aspects where performance value can be measured [110].

Calculate the Expected Benefit We can value the different strategies to estimate the future value of the system based on the level of utility in terms of compliance that can be achieved from supporting AS's. We associate utility with the different value perspectives and compliance dimensions. The total utility derived from an architectural strategy is our AS benefit equivalent. A strategy's value is determined by summing up several dimensions of utility obtained from the dimensions. The compliance dimensions that need to be protected by the compliance goal are the parameter that were specified earlier. The benefit from the architectural strategy can assume integer values from *one* to *three*.

The benefit is calculated as:

$$Gv = V_1 + V_2 + V_3 + \ldots + V_N$$

Where

1. Gv is the total value (benefit) of the architectural strategy
2. V_1 is the utility value associated with the first dimension
3. V_2 is the utility value associated with the second dimension
4. V_n is the utility value associated with the nth dimension

This can be used in place of CBAM equations. The method can benefit from the overall principle of applying CBAM [137]. We can determine the return on an architectural strategy using CBAM

$$Benefit\ \ (Gv) = \sum (contribScore)(Qattrib) \tag{1}$$

$$Return\ \ (Gv) = \frac{Benefit}{Cost} \tag{2}$$

Adjust the Value for Interdependencies The Goal Value should be adjusted based on the existing interdependencies of goals on each other. Dependency between two goals means that there is some link between them, i.e. satisfying one goal affects the achievement of the other goal. Goal Dependency is used to model the relationships that exist between goals in the system. Different types of dependencies and interdependencies exist between requirements. *Dahlstedt* and *Persson* [67] have identified several of the ways in which requirement interdependency may manifest. These are structural and cost/value dependencies, which include *Requires*, *explains*, *conflits _with*, among others. These interdependency types were analysed and combined to create the five primary interdependency factors used in this paper. An interdependency factor helps to identify and characterise the dependency between the goals.

Understanding the goal's interdependency factors and their characteristics is critical to measure accurately how much these goals depended on other goals. Each of the dependency factors represents the strength of the dependency that exist between the associated goals. The evaluation model will use five interdependency levels. These levels have values ranging from -1, which is a negative correlation to 3, which is the strongest positive relationship. The levels are used to determine the strength of the interdependency between goals. Ideally, the goal value needs to be adjusted based on the type of dependency that a goal has on another; otherwise, obstacle resolution mechanisms may not be properly implemented. Table 5.1 summarises these interdependency types.

For simplicity, we will only model the "*requires*" and *"explains"* dependency relationships. If a goal G_1 requires on another goal G_2 to be satisfied, the value of G_1 may be adjusted as

$\mathrm{Gv}_1 = \max\{ Gv_1, Gv_2\}$ (3)

Table 5.1: Interdependency Types

Level	Type
Level 3	Requires - The fulfilment of one goal depends on the fulfilment of another goal. It can happen when a goal needs another goal.
Level 2	Explains - A general goal is explained by a number of more specific goals. If a detailed goal is derived from a high-level goal but it is not a prerequisite for this goal, the relation is of the dependency type explain.
Level 1	Relates_To - A goal has a relationship with another goal. It is indicated in the literature that a goal may affect or influence another goal in other ways than requires, explains and conflicts
Level 0	Independent - The goals have no dependency connection between them.
Level -1	Conflict_With - A goal is in conflict with another goal if they cannot exist at the same time or if increasing the satisfaction of a goal decreases the satisfaction of another goal

5.4.2 Quantify Cost Implications

The previous step calculated the benefits for each strategy with compliance dimensions in mind. Under CBAM, the next step is to elicit or calculate the expected cost of implementing each strategy. We follow similar principles to that of CBAM but benefiting from the relative dimensions and metrics.

For estimating the cost, three possible routes can be pursued:

1. Use expert knowledge to cost estimation
2. Use parametric models to cost estimation
3. Combine expert knowledge with parametric models for better estimation.

Use Expert Knowledge to Cost Estimation Expert knowledge is a non-model estimation technique that relies on direct estimation. It does not rely on estimation models. Using expert knowledge requires the involvement of experts, their previous experience, and judgement to generate an estimate of the cost of an architectural strategy. Using only non-model estimation methods may lead to inaccurate results. Estimating costs using expert knowledge is suitable for projects that are not too different from previously completed projects, as the analyst may have experience in similar situations. This method is subjective and the cost estimation process is vague, that make it harder to justify the estimates.

Use Parametric Models to Cost Estimation This method uses models that constructively predict the cost of developing a software product. Model-based parametric-based estimation is usually dependent on a number of inputs to estimate costs or effort. It is very difficult to develop parametric models that yield high accuracy for software development in all domains.

5.4.3 Quantify Risk Implications (Risk Analysis)

In this step, we extend CBAM by evaluating and quantifying the risks associated with each strategy. We also evaluate how an architecture design decision can pose risks on sustainability and its dimensions. We make the link to risk explicit using portfolio thinking. After computing the goal values and adjusting the values based on their interdependencies, the obstacles to these goals and the corresponding risk values, needs to be modelled.

The term risk is defined as the potential future harm that may arise due to some present actions. In the context of compliance evaluation software, risk is defined as the probability of occurrence of unacceptable outcomes, generally caused by non-adherence to compliance goals and conflicts with other goals. Our framework uses goal–oriented requirements engineering for specifying the compliance requirements. In this framework, we use the obstacles associated with a goal to model and reason about risks. While goals capture the objectives to be satisfied, obstacles capture undesired properties that may prevent the goal from being satisfied [120] An obstacle obstructs a goal if the obstacle negates the goal in the domain[178]. Our risk analysis approach using obstacle analysis to quantify consists of two steps: obstacle identification and obstacle analysis.

Obstacle Identification The objective of the obstacle identification step is to analyse potential sets of undesirable characteristics of the architectural strategy that may prevent the satisfaction of the compliance goal. During the specify goals phase obstacles are generated from goal specifications [120]. The obstacles can be refined sub-obstacles in order to identify precise risks. Letier et al. suggested that obstacles should be identified from the terminal goals that are assigned to agents [120]. It is essential that the set of identified obstacles to the compliance goal is identified for every architectural strategy.

Obstacle Analysis To analyse the impact of obstacles, we use the obstacle analysis technique described by Ojameruaye [25]. The technique is a quantitative approach to rank obstacles using the concept of portfolio thinking and technical debt. Once obstacles have been identified, they need to be assessed and prioritised. We assert that the risk value of an obstacle is the product of the likelihood of the obstacle occurring and its criticality. We describe an approach for allocating resources for resolving obstacles as well as selecting the obstacle resolution tactic by considering the amount of technical debt that each obstacle may incur and the interest that might accumulate as the deciding factors. Consider a compliance goal that has been specified and its obstacles have already been identified, the obstacles can be formally modelled using two properties – the criticality and the likelihood of the obstacle occurring.

The criticality of an obstacle indicates how bad the obstacle is. It is determined by how much the obstacle affects the goal. The criticality of an obstacle also depends on its impact on higher-level goals and the relative importance of those goals. Every obstacle incurs a cost for its resolution (mostly resource consumption), and this affects the obstacle's criticality. In the proposed methodology, criticality is computed on a 5-point scale as shown in table 5.2 below.

Likelihood denotes the probability that the obstacle will occur. In our framework, we calculate the likelihood of an obstacle occurring using historical data and based on the value of evidence that increases and reduces the likelihood. The likelihood is defined quantitatively and can take the following values: Almost Certain (5), Likely (4), Occasionally

Table 5.2: Criticality of Obstacles

Criticality	**Interpretation**
Very High (5)	The obstacle has a high impact on multiple higher-level goals. It's impact can be catastrophic
High (4)	The obstacle has a high impact on a single higher-level goal.
Medium (3)	The obstacle produces a fair contribution to goal negation
Low (2)	The obstacle has a low impact on multiple higher-level goals.
Very Low (1)	The obstacle has a low impact on a single higher-level goal. It's impact can be insignificant

(3), Rare (2), and Unlikely (1). The likelihood can be determined by past occurrences, which represents previous incidents that have occurred due to the obstacle.

$$O_v = \log_2 (P_O * \ L_O * \ C_O) \tag{4}$$

The reason for taking logarithm to base two is to normalize the result within a scale of 0 to 5. Every obstacle has a risk factor. We can prioritise the obstacles by quantifying the risk value of the obstacles. The risk factor of an obstacle is the product of the likelihood and the criticality. Where Ov is the risk value of the obstacle, P_O is the cost of resolving the obstacle, L_O likelihood that the obstacle will occur and C_O is the criticality. Another lightweight technique to support this process consists of using the standard risk analysis matrix in which the likelihood and criticality is estimated.

Sometimes, obstacles to satisfying the compliance goal may be interdependent on other obstacles. If a goal has obstacles O_{B1} and O_{B2}, it might happen that O_{B2} can only be resolved if O_{B1} has been resolved. Consequently, resolving O_{B2} is dependent on resolving O_{B1}. This may hold for obstacles to a single goal as well as to different goals. Dependencies among obstacles are similar to dependencies among goals and can be modelled in a similar way.

Agents Liability An agent is defined as components, which are capable of performing operations to satisfy the goal. In this method, an architectural strategy is considered a type of agent. An agent will have many obstacles relationships. To compute the liability of an agent A, a list of obstacles *[*$O_1,\ O_{2,}\ \ldots\ \ O_n$*]* associated with that agent are obtained along with their obstacle values. An agent's liability is modelled as an average of the total obstacles values associated with that agent.

$$A_V = Roundoff\left(\sum \frac{Obstacle\ Value)}{n}\right. \tag{5}$$

Compliance Concern and Risk Value The compliance concern of an architectural strategy (AS) is defined as a function of the obstacles and agents associated with a goal. This is obtained by identifying the obstacles associated with that AS that can obstruct the satisfaction of compliance goals. The compliance concern is obtained by identifying the most critical obstacle, so the concern for an AS is the maximum of all the obstacle values for that AS as given below. If no obstacles exists, then the value is assumed zero

$$C = \max(O_{v1,}\ O_{v2,}\ldots,\ O_{vn} \tag{6}$$

Finally, the compliance risk factor for the AS is computed as the logarithm of the product of the benefit and compliance concern. This computes and presents the risk value associated with the AS.

$$Rv = \log_2(Gv * C\ ) \tag{7}$$

Risk factor provides a summary value of risk for an AS. Risk values for all identified obstacles should be computed by the method described above.

5.5 Phase III: Portfolio Analysis

We calculate a desirability metric by which the various ASs that have passed the triage stage can be compared. In CBAM, this is done with a ranking metric, which calculates desirability as a function of benefit and cost. This does not account for risk, hence, our use of portfolio thinking.

We now analyse the costs, benefits and risks associated with alternative decisions to create an architectural design using portfolio theory. Since non-functional requirements do not have simple true or false satisfaction criteria but are satisfied up to a level(A Lamsweerde 2004), we can determine how well the goal needs to be achieved with the available resources. With the measurements of the value of the individual options as described above, inputs to the portfolio approach can be estimated and we can start making decisions using the portfolio approach. Each AS has a risk value R_1, a cost P_1 and W_1 as the weight of the risk. Based on these values, we can then decide on how many instances of the risks to the goal need to be resolved so that global risk of the goal being obstructed is reduced.

One can see that architecture has similarities to investment financial market with the view of diversification to reduce risks and maximise returns. However, we use portfolio theory to identify an architecture as optimal portfolio of architectural strategies with a total value that minimises risks and maximises returns.

Where total cost of an architecture is

$$C = \sum_{i=1}^{m} C_1 \tag{8}$$

And Weight W is

$$W = \frac{C_1}{\sum_1 C_1} \tag{9}$$

Where P_1 is the return on the architectural strategy and W_1 is the weight assigned to that strategy, the expected return E_p of an architecture is

$$\int_{i=1}^{n} \mathrm{Ep} = 1 \tag{10}$$

$$E_p = \sum_{i=1}^{m} W_1 \left(\frac{0B_1}{C_1} \right) \tag{11}$$

$$E_p = \sum_{i=1}^{m} W_1 P_1 \tag{12}$$

Where R_1 is the local risk of the architectural strategy and W_1 is the weight assigned to that strategy, the global risk of an architecture is

$$R_p = \sum_{i=1}^{m} \sqrt{W_1^2 R_1^2} \tag{13}$$

Evaluating and selecting the best architecture based on the requirements is a core activity in the requirement elaboration process. The next step is to shortlist valid and optimal architectures. An architecture is valid, if it satisfies the relevant constraints such as cost, risk, schedule or compliance score thresholds. The ideal plan is to shortlist valid architectures that maximise expected net benefits and minimise risks on dimensions related to compliance.

Identifying the optimum architecture is a core activity in the evaluation process. An architecture is optimal for a given problem if it achieves minimum risk and maximises value, while satisfying critical constraints. This is defined as a portfolio optimisation problem to minimise risk and maximise return subject to cost constraints. Where E_R is the expected return and P_d is the standard deviation of the portfolio, we define this in terms of Sharpe ratio:

$$\textit{Maximise}\ \mathrm{E_p/P_d} \tag{14}$$

Subject to lower limit $\leq$ *Cost* $\leq$ *upper limit*

This can also be used for comparison and benchmarking of other architecture portfolios. The architecture with a high Sharpe ratio is the one that achieves the best combination of value, risk in presence of uncertainty.

5.6 Phase IV: Debt Analysis

Evaluating and selecting the best architecture based on the requirements is a core activity in the requirement elaboration process. We evaluate the architecture by considering the amount of compliance debt that each architecture may incur as the deciding factor.

The sharpe ratio can also be used for comparison and benchmarking of other architecture portfolios. The architecture with a high Sharpe ratio is the one that achieves the best combination of value, risk in presence of uncertainty.

The ability to find the optimal solution is complemented with the ability to rank architecture candidate portfolios. The architecture is a portfolio(s) of ASs, where each AS has its risk, and benefits on goals and the compliance dimensions. We rank the architecture (i.e. portfolio for our case) by considering the amount of debt that each architecture may incur as the deciding factor. We calculate and assign the compliance debt of each alternative. We evaluate the different architectures by considering the amount of compliance debt that each architecture may incur as the deciding factor. From our earlier explanation of the debt using the technical debt metaphor in as the gap between what can be achieved with the available resources and the hypothesised "ideal" environments where the all compliance goals are successfully achieved. We formulate the value of the debt:

$$Maximise \ \mathrm{E_p}/\mathrm{P_d} \tag{15}$$

Subject to lower limit $\leq$ *Cost* $\leq$ *upper limit*

$$T_D = I_{RT} - R_T \tag{16}$$

Where T_D is the debt, I_{RT} is the Sharpe ratio of an "optimal" architecture and R_T is the Sharpe ratio cost of the selected architecture. The ideal value is context dependent. The ideal value is application and business dependent. T_D for any other tactic is calculated as the gap between value of architecture k (I_{RT}) and the value of the architecture in question. The architectures can then be ranked from best (low T_D) to worst (high T_D).

This approach models technical debt as a function of requirement trade-offs. This approach presents the optimal solutions at a given point in time in terms of specific compliance and value requirements. Even though we may implicitly know that a particular architecture is not optimal in the long-term, presumably we choose this architecture it meets the objectives at that point in time. It is only when time passes that debt begins to accrue as the compliance and business needs begin to favour the optimal solution more than the actual solution. With respect to "interest" on the debt, which we loosely translate as the cost of neglecting the debt.

The debt can be attributed to the inability of the architecture to partially or fully satisfy compliances requirements. We claim that a compliant system tend to add value with time and as it evolves and the added value is relative to the stakeholders involved.? Debt can be relative to the scenarios. It could be discussed from the perspective of the end users? and the perspective of the provider (system developer). When TD is discussed in conjunction and relative to all the scenarios of interest, it can give an indication of the overall system's sustainability. Compliance debt identified from this evaluation can be attributed to different situations such as

1. The architecture does not fully meet the requirements

2. Overdesign of the architecture so the potentials of the architecture is not fully utilized and the operational cost tends to exceed that of the generated benefits

3. Absence of publicly available historical data for analysing impact of decisions that may lead to poor and quick decisions. These may that add a value in the short-term but can introduce long-term debt in situation where improving the system is unavoidable.

5.7 Phase V: Interpretations and Recommendation

The final stage of the method is the evaluation and interpretation of the results relative to the set objectives. This phase consists of incorporating all information obtained during the earlier phases and using them to inform the decision process. The ultimate objective of this phase is to choose the "best solution". Firstly, let us understand what constitutes a "best solution". By "best", we mean that the selected architecture does not need to be optimal but satisficing. A satisficing architecture is the one that sufficiently satisfies the set of goals defined by stakeholders; while an optimal architecture aims, at any cost, to maximise the satisfaction of all goals. Here cost represents monetary investment, time, effort, complexity, or anything that may constrain the full satisfaction of goals. Given the complexities, investments and challenges involved in software development, it is reasonable to say that selecting the optimal architecture for development is not feasible.

As illustrated in Figure 5.8 , the first proposed scenario explores some fundamental issues to be considered to judge the worth of each architecture. The value axis is assessed by the overall value and satisfaction score of the solution being examined. The risk axis is assessed through obstacle analysis. Finally, the cost axis is measured in terms of involved costs to deploy the solution. The in-depth examination of all these interdependent issues involved in the selection of a particular architecture, allows the evaluation team to make rational decisions that are justified by well-defined arguments.

In order to facilitate the identification of satisficing architectures or solutions, we use the debt that will be accrued by each of the architecture based on a combined measure of the debt, cost, risk and value presented in each alternative architecture. The suitability of each alternative is measured in terms of these three constraints, where the satisficing architecture will be a compromise among, risk, debt and cost. At this stage, however, it is clear what are the cost and risk constraining the satisfaction of the goal. It is also clear the amount of debt that will be incurred by each alternative architecture. As a result, the evaluation team is able to choose appropriate architecture with its associated trade-offs

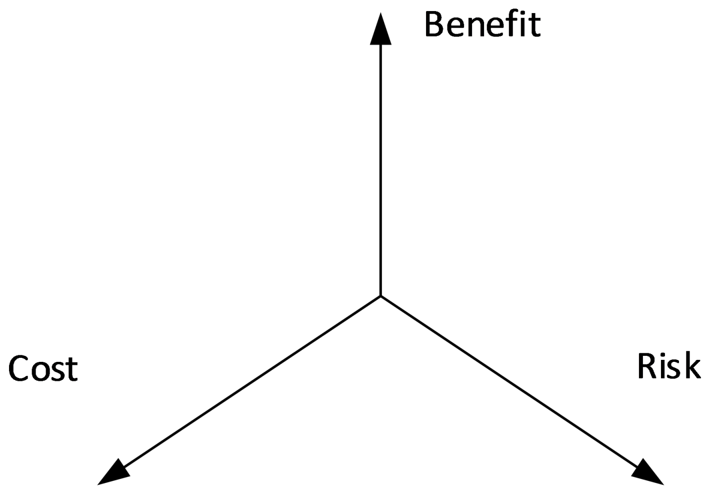


Figure 5.8: Factors to assess the worthiness of each architecture

visualised as a compliance debt to accommodate such constraints. At this stage, earlier decisions have to be reassessed. This phase finishes when the evaluation team makes the final decision to select the architecture that brings the higher overall value, where value refers to the relative worth of the architecture with respect to the stakeholders' objectives and constrains.

The supporting method is open and flexible enough to address the objectives. The method does not define rigorous or prescribed actions to follow. Although the steps are numbered suggesting linearity, this is not a strict waterfall process. There will be times when an analyst will return briefly to an earlier step; will jump forward to a later step; or will iterate among steps, as the need dictates. Furthermore, the analyst may amend the steps based on the available information at hand, the case itself, and the set evaluation objective(s). Accordingly, the nature of the decisions due to the application of the model fundamentally varies with the nature of the problem, across projects, and organizations. As a result, such decisions are subject to the objective for which the model/method is applied. Briefly, the recommendations are tailored to the set evaluation objective(s).

5.8 Summary

In this chapter, we developed the method for addressing compliance requirements at the architectural level. The method uses a goal-oriented approach for eliciting and modelling the compliance goals. The methods relies on CBAM and portfolio thinking to facilitate the analysis and operationalisation of compliance requirements. We have presented systematic guidance for the identification and evaluation of compliance requirements. We have discussed issues related to conducting these steps, as it was realized in the application of the method. The method does not prescribe rigorous steps to follow. It aims to discuss issues and provide ways for estimating the necessary valuation parameters. The framework can incorporate multiple valuation techniques. We have explored ways for estimating the parameters in the context of use.

A key phase of the method is the debt analysis process. CBAM is used to model the benefits (satisfaction function) of the architectural strategies related to the goals. This measures the degree to which strategies satisfy the compliance requirements. We use portfolio thinking to identify an architecture as optimal portfolio of architectural strategies with a total value that minimises risks and maximises returns. The ability to find the optimal solution is complemented with the ability to rank architecture candidate portfolios and calculate the compliance debt of each alternative.

In chapter 6, we will explore cases that highlight possible application of the model and its supporting method.

CHAPTER 6

EVALUATION

”Fear cannot be banished, but it can be calm and without panic; it can be mitigated by reason and evaluation”

Vannevar Bush

In previous chapters, we have described a model for evaluation and prioritising compliance requirements with portfolio thinking. We have supported the model with a five-phase method. In this chapter, we evaluate the feasibility and usefulness of the method using case studies.

6.1 The Evaluation Method

We evaluate our work using case studies to demonstrate the concepts and techniques used in this thesis. It is difficult to conduct controlled and repeatable experiments in software engineering [151]. Although case studies have no clear theoretical basis from which to draw strong conclusions, it is widely believed that a sound case study produces meaningful results in the software engineering domain [113]. Case studies have been used to assess software engineering approaches [151]. Evaluating methods like ours is rather hard, as their effectiveness is extremely dependent on the way they are applied. Software engineering methods are generally tailored to address the needs of specific projects and suit different contexts. This means that some projects may not need all the prescribed actions, artefacts and guidelines defined in the method, while others need to follow all the activities rigorously. As a result, the validation of the method is subject to the context in which the method is being applied. While the results obtained from the evaluation effort presented in this chapter demonstrate the suitability of the method for the classes of problems in which the method has been examined, it is not possible to guarantee that it is suitable for other projects.

The case study was conducted following the relevant guidelines for software engineering tools and methods [113]. The DESMET methodology [111] provides hints for guiding the evaluation of software engineering methods.

To establish the effectiveness of the method we have conducted two forms of evaluation:

1. A real-life case study

2. Simulated case study from literature

In order to control the results of the evaluation effort, we conducted the case studies according to the DESMET methodology [111] to guild the evaluation of software engineering methods. According to [111], the first decision to make when undertaking a case study is to determine what the study aims to investigate and evaluate. This evaluation aims at exploring the method's effectiveness in addressing desired compliance requirements. The evaluation aims at demonstrating the method's applicability and simulating the models' application. For evaluating our method, the goal we want to achieve is to show the validity of the following hypothesis:

"The proposed method is the systematic way to evaluate compliance requirements and aid decision-making that is value and risk aware".

In order to test the validity of the stated hypothesis, we defined the qualities we expect the method to have. We explicitly define the criteria under which we will evaluate the results of the case studies:

1. Is Goal-based requirements engineering a suitable approach to specify compliance requirements?

2. Is the risk analysis strategy using obstacle analysis effective in identifying and analysing risk that may arise during the evaluation process?

3. Does the economics-driven model that exploits portfolio theory provide insights into allocating available resources in a manner, which will minimize costs, reduce risk and maximize value?

4. Does the model exploit the concept of technical debt to provide a metric for visualising trade-offs between short-term and long-term value in compliance?

5. Does the method provide complete, understandable and useful method to guide the evaluation of compliance requirements?

DESMET has identified nine different evaluation methods [111]. We are using the "qualitative screening" method of evaluation which is a feature-based evaluation done by a single individual, who decides the features to be assessed and carries out the assessment [111]. We evaluate method on some qualitative characteristics including simplicity of use, correctness, openness, and comprehensiveness. We reflect on strengths and limitations of the method upon conducting the case studies.

6.2 Case Study 1 - Emergency Deployment System (EDS)

This section demonstrates of the viability of the framework through the case study of a software system called Emergency Deployment System (EDS). It describes the related compliance requirements and how the framework can be used to analyse and aid decision-making. Through the case study, we explore whether the framework presented in this thesis does add value to requirement engineering methods.

The case study was selected from emergency response as the systems in this domain need to be sustainable to accommodate uncertain conditions, increase efficiency, reduce cost with minimum risk to people. There is a strong correlation between the accuracy, timeliness, and reliability of the information available to the decision-makers, and the quality of their decisions.

The case study was selected from emergency response as the systems in this domain need to be sustainable to accommodate uncertain conditions, increase efficiency, reduce cost with minimum risk to people. EDS consists of two subsystems: Headquarters and Search and Rescue. The Headquarters subsystem manages several Search and Rescue subsystems. The Search and Rescue subsystem is a mobile application that runs on smart-phones and tablets, and used by emergency crew members. The original subsystem was relatively simple, providing support for sharing data with Headquarters. The recent propagation of technologies presented an opportunity to improve this part of the system. Consequently, a decision was made to re-architecting the system. The new application was intended to allow the crew to share and obtain an assessment of the situation in real-time, coordinate with each other, and engage with the Headquarters.

In re-architecting the EDS application, a team was formed, to decide among the early design decisions. These decisions are shown in Figure 5.8. Each decision consists of several viable alternatives. We will also introduce a hypothetical goal of complying with sustainability requirements for the system.

6.2.1 Rationale

Worldwide, sustainability regulations are being strengthened. Complying with sustainability requirements need to be aligned with the systems, enterprise, software goals and prevailing regulations to control compliance, to add value, and to meet the needs of the business. With recent developments, some software systems have some sustainability requirements to which it should comply with to retain or improve its value. On one hand, the stakeholders are interested in maximising returns of the systems while minimising risks and cost of development.

Sustainable development can be defined as “the ability to meet the needs of the present without compromising the ability of future generations to satisfy their own needs” [185].

Thinking about sustainability in software systems is important as the on-going use of software systems may involve new burdens on social and environmental systems [26]. Software-intensive systems should be environmentally, socially, economically, individually and technically sustainable [24] [149]. The system should be compliant to sustainability requirements. Currently sustainability requirements are treated can be essential compliance requirement for long lived software. Worldwide sustainability requirements are being strengthened and treated as a requirement for compliance in some domains. There has been a push for government policies to on sustainability to tackle global challenges.

We apply the method to a case study from the literature [74]. The main objective of the study is to simulate the each phase of the method in detail in order to demonstrate the method's applicability of the models and validate its interpretations.

Considerations for the Case Study The case study was undertaken in a research environment. Knowledge about the domain was developed through an industrial and scientific literature review about the emergency search and rescue process. The framework modelling process is conducted by the author of this thesis. The description of typical processes employed in 21st century emergency search and rescue systems was developed through study of [11].

The process selected for evaluating the framework was re-architecting the EDS application to achieve the system's goals. In addition, the focus was on the interpretation of sustainability compliance requirements and the impact on design decisions.

We acknowledge that many factors differ between conducting the case study in a research and an industrial setting. However, we focus in the case study on addressing the objectives introduced in section 6.1 on compliance analysis using the framework. This will allow us to identify problems based on the demonstration within the context of the case study. The investigation of fundamental relationships of all the interacting factors in the case study and the research are beyond the scope of this thesis.

6.2.2 Methodology and Case Study Artefacts

The specific objective of this case study was to investigate how method can be applied to aid compliance to sustainability requirements. Though this case study was not conducted on an actual project, which means that the results may not be a conclusive account of the method effectiveness. We still consider that the knowledge gained by conducting this simulation case study represents an important piece of evidence [111] regarding the applicability of the method. This case study is presented from the evaluation team's point of view. In the next sections, we follow each one of method's phases in order to evaluate the relevant requirements.

Phase I: Identification Specify goals

To start applying the method, we had to understand the key objectives of the project. The scope, objectives and constraints of this case study were captured through a literature review. This rigorous review helped the author to identify the business and organizational concerns related to the system.

The strategic requirement related the sustainability compliance requirements are listed in Table 6.1:

Table 6.1: Compliance Requirements for EDS

1.	Minimise cost
2.	Minimise battery usage
3.	Maximise reliability
4.	Minimise response time
5.	Minimise ramp up time
6.	Minimise development time
7.	Minimise deployment time

The functional goals of the system are shown in Table 6.2

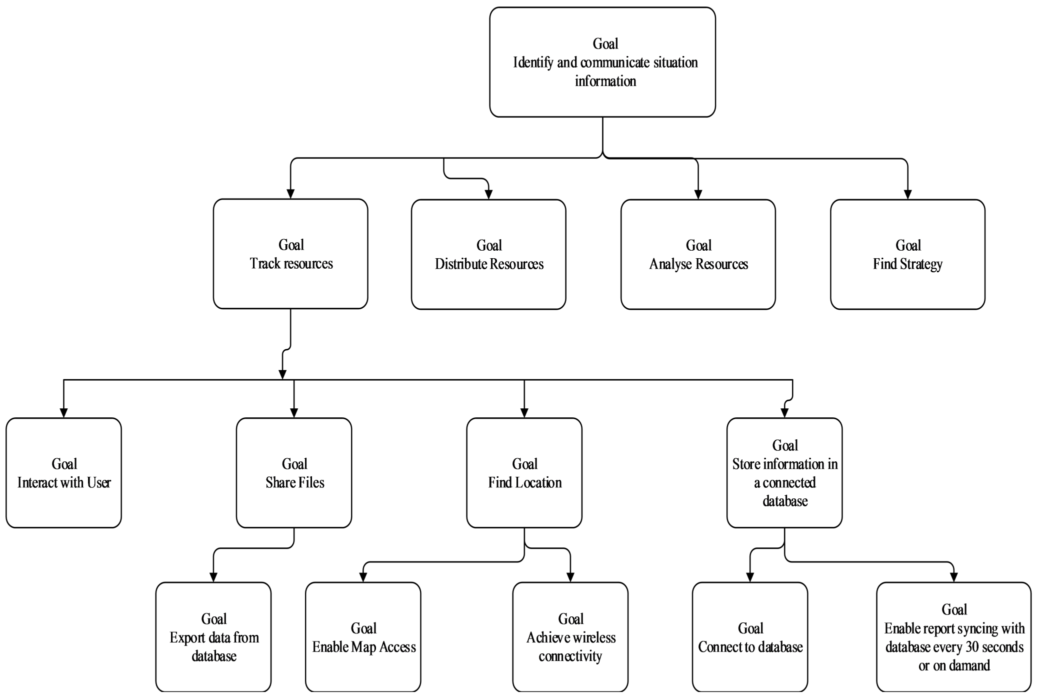
The relevant regulatory goal is

1. The system should be compliant to sustainability requirements.

The important objectives related to the compliance goal are to minimize battery usage and cost as well as improve reliability. Figure 6.1 shows the elaborated goal model

Table 6.2: Functional Goals of EDS

1	Achieve [The system should support location finding]
2	Achieve [The system should support File sharing]
3	Achieve [The system should support Report syncing]
4	Achieve [The system should support Chat protocol]
5	Achieve [The system should support Map access]
6	Achieve [The system should support Hardware platform]
7	Achieve [The system should support Connectivity]
8	Achieve [The system should have a database]
9	Achieve [Architectural pattern]
10	Achieve [Data exchange format]



Goal Identify and communicate situation information
Goal Track resources
Goal Distribute Resources
Goal Analyse Resources
Goal Find Strategy
Goal Interact with User
Goal Share Files
Goal Find Location
Goal Store information in a connected database
Goal Export data from database
Goal Enable Map Access
Goal Achieve wireless connectivity
Goal Connect to database
Goal Enable report syncing with database every 30 seconds or on damand

Figure 6.1: EDS Elaborated Goal Model

Table 6.3 shows a summary of the requirements and the alternative architectural strategies for satisfying the requirements.

Phase II: Modelling to Assist Valuation Identity and Quantify Benefits

In this case study, we use the utility an architectural strategy contributes to a compliance dimension for valuing its benefits. This quantifies the contribution of the architectural strategy to the "compliance goal" satisfaction We extend classical CBAM by assessing the utility of the architectural strategy with respect to its impact on the sustainability dimensions. Once the scenarios have been developed and their values elicited,

Table 6.3: Summary of Requirements for Case Study

Requirements	Alternatives	Related Quality/ Sustainability goals
Location finding	1. GPS	1. Minimise cost
	2.Radio transmission	2. Minimise battery usage
Hardware Platform	Nexus 1 (HTC)	3. Maximise reliability
	2. Driod (Motorola)	4. Minimise response time
File Sharing Package	1. File manager	5. Minimise ramp up time
	2. In house	6. Minimise development time
Report Synchronisation	1. Explicit	7. Minimise deployment time
	2. Implicit	
Chat Protocol	1. Openfire	
	2. In-house	
Map Access	1. On demand (google)	
	2. Cached (google server)	
	3. Preloaded (ESRI)	
Connectivity	1. Wi-Fi	
	2. ÂčG on Nexus	
	2. 3G on droid	
	4. Bluetooth	
Database	1. MySQL	
	2. SQLite	
Architectural Style	1. Peer-to-peer	
	2. client -server	
	3. Push-based	
Data Exchange Format	XML	
	Compressed XML	
	Unformed Data	

their relationship to the different sustainability dimensions is determined. The degree to which each of these attributes is expected varies from domain to domain and tends to be benefit from expert's input. The evaluation benefiting from these dimensions shall be long-term and strategic in nature. For complying to sustainability, we use the sustainability dimensions described in [24]. These are the individual, economic, environmental, technical and social dimensions.

As said in chapter 5, We use the Software Value Map [110] to identify the different value perspectives that can be used evaluate the impact of an architecture strategy on these dimensions.

1. The financial perspective

2. The customer perspective

3. The internal business process perspective

The table 6.4 shows the interrelationship between different value perspectives, the compliance dimensions and indicative value indicators that the analyst can consider when valuing for sustainability compliance requirements.

Quantifying the benefits of architectural strategies

We are interested in understanding the utility relating to compliance that can be derived from an AS. We quantify the impact on the compliance dimensions. Given an alternative for satisfying the goals, we are interested in understanding its impact on complying with sustainability requirements. We estimate the impact of a strategy on compliance using the already elicited parameters. For example, battery usage can be linked to both complying with the environment and technical sustainability requirements; this is because an architecture with a high battery usage may lead to low availability. Consequently, the solution may not be technically sustainable in the context of emergency response systems. In addition, a high battery usage means an increased carbon footprint and this inversely affects environmental sustainability. Similarly, reliability and response time figures could be linked to technical and individual sustainability, where any degradation in reliability may have negative consequences. Increased development and deployment times could be linked to increased carbon footprint, which can negatively affect compliance on the environmental sustainability dimension.

We use the following value indicator for this case study, derived from table 6.4. We consider the below metrics for H, Ec, T and En for simplicity.

1. Individual (H) – measured by the number of people- hours involved in development and implementation

2. Economic (Ec) - measured by the total development and implementation costs

3. Technical (T) - The capability of the feature to maintain a specified level of performance when used under specified conditions (includes maturity, fault tolerance and

Table 6.4: Sustainability Dimensions, Value Perspectives and Value Indicators

	Financial	**Customer**	**Internal process**
Individual	The financial implications of user satisfaction and retention	Sustained perceived value to the user (e.g., perceived satisfaction, convenience, etc.).	Implications on resources including human, time, etc. in relation to development, maintenance and evolution etc.
Economic	Total development, maintenance and evolution costs among the others	Cost and value in supporting user's specific concerns (e.g. can relate to users' specific functional and non-functional requirements).	Cost, value and risks for developing for, maintaining and monitoring sustainability
Environmental	The financial implication of complying and/or non-compliance for environmental sustainability. Market losses or gains relative to market sustainability.	Environmental impact due to energy usage and CO2 emissions	Meeting market and compliance requirements- their cost. Risk and value implications.
Technical	Total development, maintenance and evolution costs among the others	cost and value in supporting functional and non-functional requirements along	The value of maintaining the system over time
Social		Value from incentives such as cost, complimentary value through network of users and technologies	Development team management

recoverability)

4. Environmental (En) – observed through energy usage.

Usually qualitative metrics may be extended by quantifying or replacing value such as high, medium or low with numerical weights. Although this principle is often used in practice, problems such as lack of precision (for precise requirements) and lack of clear criteria to validate the estimation still exists, as this technique is subjective. For simplicity, we propose quantitative measurement metric based on subjective criteria such as the stakeholders' experience. The parameters value range from one to three is assigned depending on if the impact is high, medium or low. The decision on which level the parameters used, depends on the scope of the system under analysis. We compute the impacts values using a scale of 1 to 5 using three point estimates for each parameter (most likely, lowest and highest values).

Table 6.5: Portion of Sustainability Compliance Analysis

Attribute		Human	Economic	Technical	Environmental
Description of value indicators		Number of people hours involved in development and implementation	Total development and implementation costs	The capability of the feature to maintain a specified level of performance when used underspecified conditions (includes maturity, fault tolerance and recoverability).	Energy usage
Location finding	GPS				
	2.Radio transmission	2.12	0	4.8	1.7
Hardware Platform	Nexus 1 (HTC)	0	3	0	1.4
	Driod (motorola)	0	3.2	0	3.1
File Sharing Package	1. File manager	1.68	0	4.9	1.9
	2. In house	1.74	0	4.6	1.6
Report Synchronisation	1. Explicit	0.89	0	4.5	0.5
	2. Implicit	0.58	0	5	3.4
Chat Protocol	1. Openfire	1.39	0	4.9	1.7
	2. In-house	1.76	0	4.3	2.1
Map Access	On demand (google)	2.39	0	4.3	2.6
	Cached (google server)	2.78	5	4.8	2.8
	Preloaded (ESRI)	3.83	0.9	4.7	2.4
Connectivity	1. Wi-Fi	0.96	0.5	4.5	1.2
	2G on Nexus	0.57	2.7	4.6	0.2
	3G on droid	0.57	2.7	4.6	1
	Bluetooth	1.29	0.6	3.8	2.6
Database	MySQL	2.73	0	4.7	1.9
	2. SQLite	3.33	0	4.4	2.6
Architectural Style	1. Peer-to-peer	5	0	3.4	3.4
	2. client -server	3.15	0	4.8	2.2
	3. Push-based	3.92	0	4.9	0.9
Data Exchange Format	XML	0.89	0	0	1.4
	Compressed XML	1.28	0	0	2.1
	Unformed Data	0.55	0	0	0

Quantify Cost Implications

The cost implications, shown in Table 6.6 were elicited as part of the original case study.

Table 6.6: Cost implications

Requirements	**Alternatives**	**Costs**
Location finding	1. GPS	0.5
	2.Radio transmission	0
Hardware Platform	Nexus 1 (HTC)	3
	2.Driod (motorola)	3.2
File Sharing Package	1.File manager	0
	2. In house	0
Report Synchronisation	1. Explicit	0
	2. Implicit	0
Chat Protocol	1. Openfire	0
	2. In-house	0
Map Access	1. On demand (google)	0
	2. Cached (google server)	5
	3. Preloaded (ESRI)	0.9
Connectivity	1. Wi-Fi	0.5
	2. 3G on Nexus	2.7
	3. 3G on andriod	2.7
	4. Bluetooth	0.6
Database	1. MySQL	0
	2. SQLite	0
Architectural Style	1. Peer-to-peer	0
	2. client -server	0
	3. Push-based	0
Data Exchange Format	XML	0
	Compressed XML	0
	Unformed Data	0

Quantify risk using obstacle analysis

In our methodology, we define risk as a function of the obstacles to a goal and the liabilities incurred by architectural strategy with respect to satisfying the compliance goals. After computing the benefits and adjusting the values based on interdependencies, the obstacles to these goals and the corresponding risk values, needs to be modelled. Obstacles can be modelled using two properties – the criticality and the likelihood of the obstacle occurring.

Using goal model produced in the previous step, the compliance goal-obstacle analysis consists of identifying obstacles that may prevent the compliance goal from being fully satisfied and assessing the likelihood and criticality of these obstacles.

Identifying Obstacles

We exemplify the identification of such obstacles for the following compliance goal and the architectural strategies:

Achieve [The system should support sustainable location finding].

The GPS component is responsible for the leaf goal: Achieve [The system should support sustainable location finding]. This goal is compared against a set of known sustainability goals for the domain such as reliability and battery usage. Conditions that negative these goals and at which these goals will not be satisfied are identified. For example, a component with a high battery usage will have a negative impact on the usability of the system, which in turn will negate the sustainability goal.

Repeating the process for other compliance goals and the architectural strategies, a list of obstacles identified are listed in the Table 6.7 below. For simplicity, we assume that the obstacles to the compliance requirement are the same across each goal and AS. In a more realistic scenario, the obstacles obstructing the compliance goal for each architectural strategy will be different, based on the specified level required for satisfying the goal.

Table 6.7: EDS Obstacle Analysis

Obstacle	**Criticality**	**Value**
Cost of goal exceeds the cost that is economically sustainable	High	5
Unsustainable (high) battery usage	High	4
Component is not reliable enough	High	5
Long response time	High	4
Long deployment time	Medium	3
long development time	Medium	3

Assessing Obstacles

Obstacles assessment consists of estimating the likelihood and criticality of obstacles obstructing the compliance requirements. The likelihood of the obstacles in this assessment can be classified as 'almost certain'. For this reason, we show in Table 6.8 only the estimated criticality of the identified obstacles. Alternatively, the likelihood of the obstacle is already know from the evaluating the benefits as some AS may positively influence the compliance goal more than others may.

Using the metrics and the method defined in chapter 5 , we compute the obstacle value and the risk factor for each architectural strategy.

Table 6.8: Risk Analysis - EDS

Requirements	Alternatives	Agent value	Obstacle Value	Risk Value
Location finding	1. GPS	0.41	0.8	33%
	2.Radio transmission	0.31	0.96	29%
Hardware Platform	Nexus 1 (HTC)	0.15	0.6	9%
	2.Driod (motorola)	0.21	0.64	13%
File Sharing Package	1.File manager	0.25	0.98	24%
	2. In house	0.24	0.92	22%
Report Synchronisation	1. Explicit	0.2	0.9	18%
	2. Implicit	0.28	1	28%
Chat Protocol	1. Openfire	0.25	0.98	24%
	2. In-house	0.26	0.86	22%
Map Access	1. On demand (google)	0.41	0.86	36%
	2. Cached (google server)	0.52	1	52%
	3. Preloaded (ESRI)	0.37	0.94	35%
Connectivity	1. Wi-Fi	0.24	0.9	21%
	2. 3G on Nexus	0.27	0.92	25%
	3. 3G on andriod	0.29	0.92	27%
	4. Bluetooth	0.28	0.76	21%
Database	1. MySQL	0.35	0.94	33%
	2. SQLite	0.38	0.88	33%
Architectural Style	1. Peer-to-peer	0.4	0.68	28%
	2. client -server	0.34	0.96	33%
	3. Push-based	0.34	0.98	33%
Data Exchange Format	XML	0.07	0.22	2%
	Compressed XML	0.09	0.34	3%
	Unformed Data	0.02	0.1	0%

Phase III. Portfolio Analysis With the measurements of the values for the individual strategies as described above, inputs to the portfolio approach can be estimated and we can start making decisions using the portfolio approach.

The next step is to determine the overall desirability of a candidate architecture design, which relates to a portfolio of ASs. The overall effectiveness of an architecture is a function of the total expected return and the global risk of that portfolio. Table 6.9 shows the analysis and result of a candidate architecture, from selecting specific AS.

Table 6.9: Candidate Architecture along with the Results of the Analysis

Goals	Candidate AS	Result of the Analysis	
Location finding	Radio transmission	Ep (Expected Return)	10%
Hardware Platform	Nexus 1 (HTC)	Rp (Global Risks)	77%
File Sharing Package	In house	Cost	756.78 pound
Report Synchronisation	Explicit	Sharpe Ratio	12%
Chat Protocol	Openfire		
Map Access	3. Preloaded (ESRI)		
Connectivity	Wi-Fi		
Database	MySQL		
Architectural Style	3. Push-based		
Data Exchange Format	Unformed Data		

We shortlisted valid architectures that maximise expected net benefits and minimise risks on dimensions related to sustainability. An architecture is shortlisted, if it satisfies the relevant constraints such as cost, risk, schedule or the compliance score threshold.

One of the requirements elicited by the original design team in EDS was to keep the cost of a single handheld device below \$1000. For this example, we assume that optimal architecture is the architecture with the highest Sharpe ratio while satisfying the cost constraint of \$1000. For the case study, there was 6912 alternatives and after the constraint was applied, we had about 2730 valid architectures and the optimal architecture had a cost of \$968.94, an expected return of 11% and a global risk of 68%.

The presented results are extracts from the conducted case to ease exposition. Nevertheless, the analysis of the case can benefit from automated environments to deal with the various permutations.

Phase IV. Technical Debt Analysis We evaluate the architecture by considering the amount of compliance debt that each architecture may incur as a deciding factor.

Using equation 2 from chapter 4, we ranked the other architectures from best (low S_D) to worst (high S_D). This can also be used for comparison and benchmarking of other candidate architecture portfolios. The architecture with a low debt Sharpe ratio is the one that achieves the best combination of value, risk in presence of uncertainty. The ideal architecture is context-dependent.

6.2.3 Results and Discussion

After modelling the different combinations of architectural strategies using portfolio thinking with the cost constraints, one optimal architectural design portfolio is selected. While some of the findings from the analysis may be obvious, they were not obvious prior to the evaluation of the architecture using the method and given the possible valid combinations for constructing a portfolio.

The main objective of the evaluation was to improve for compliance to sustainability by maximising the returns of architectural strategies and minimising the risks associated with architecture achieving the system's goals through portfolio. The debt metric provides insights on the significance of an architecture in mitigating risks given the resources in hand. This is calculated as the gap between the Sharpe values of architecture in question relative to the ideal architecture achieving the system's goals for resolving this obstacle. As investing in the ideal architecture is not always affordable, the metric is an expression for the risks tolerated if a different architecture is chosen. This analysis provides analysts and architects with an objective tool to assess and rethink their investment decisions in architecture using expected returns, global risk, and Sharpe ratio of portfolio value assuming validity and accuracy of the analysis and values. The use of the debt metric had made both the short term and long-term risk visible in the evaluation and selection process. In addition, the results can improve the architect's confidence in the fitness of the proposed architecture towards meeting critical compliance requirements. The assessment process is iterative and continuous. Further investigation and application of the method to an extended real case is required to confirm the validity of these claims.

The data reported in the case study are limited. We used the data from the literature where the case study was initially presented. We used subjective values, which were derived consulting similar projects in the literature. While this may lead to subjective results, the estimates are useful in providing provide insights. The evaluation and the quantification of values we used were intentionally oversimplified to focus on demonstrating the technique and reasoning.

The technique presented in this paper, does not consider at the interrelationship between the impacts on different value aspects. This is necessary because decisions on the selection of architectural strategies based on impacts on a single dimension could obscure other impacts that might cause equal or greater damage in other dimensions. The interrelationships can exist as the following effects:

1. A positive impact on one value perspective might have positive impact on one or more compliance dimension

2. A negative impact on one value perspective might have negative impact on one more

compliance dimension

3. A positive impact on one value perspective might have a negative impact on one or more compliance dimension and vice versa

The evaluation presented in this section is limited primarily by the assumed nature of the sustainability compliance requirements. We assumed these requirements are visible to software engineers and that their impacts are easily quantifiable. Usually qualitative metrics may be extended by quantifying or replacing value such as high, medium or low with numerical weights. Although this principle is often used in practice, problems such as lack of precision (for precise requirements) and lack of clear criteria to validate the estimation still exists, as this technique is subjective. For simplicity, we use quantitative measurement metric based on subjective criteria such as the stakeholders' experience. The values range from one to five is assigned depending on if the impact is high, medium or low. For the valuation to be accurate, it is important to estimate the dimensions and values accurately. In this study, our focus is more on developing a framework for quantitative analysis system rather than model parameterization.

6.2.4 Threats to Validity

This case study was based on a study reported in the literature. The objective for carrying out the case study was to provide feedback on the method. For this reason, we relied on the parameters elicited by the initial case study and made some simplified assumptions. This imposed a few threats to the validity. The threats to validity considered based Wohlin [184] are construct, internal and external validity threats.

1. Construct Validity: A threat to construct validity is that the case study was the data used for the study. The data was assumed a fair representation of the actual characteristics of the system to be built as it was elicited from stakeholders with expert knowledge. In reality, this cannot be guaranteed. In addition, the analysis was performed by the PhD candidate, who may be biased to produce a positive

result.This is considered a threat to the *construct validity* of the study. This risk was reduced by comparing the results of this study to the published results of the original study

2. Internal Validity: This study does not account for all goals and architectural strategies that may affect the system's compliance.

3. External Validity: A threat to external validity of the study was that the architecture had already been evaluated in the literature; hence, we had an idea what the optimal architecture should be. In the analysis of a new system, the deriving the parameters for the models may not be as straightforward or easy. This risk is prevented as the evaluation method described in chapter 5, which provides a systematic process for quantifying the relevant parameters.

6.3 Case Study 2 – Security Incident and Event Management System

SIEM combines Security Information Management (SIM) and Security Event Management (SEM). The first focuses on analysis and reporting of log data and long-term storage while the second focuses on real-time monitoring and notifications.

The possible application of the methodology proposed in this thesis is demonstrated. In this thesis, SIEM is observed from the perspective of a large university that is planning to implement a SIEM as part of their IT security landscape. The methodology is based on the research at that organization. WE are unable to disclose the details of the organisation as the implementation is still ongoing.

The company wants to implement a SIEM solution and they are considering open source. The best thing about being open source is that theoretically the system will be fully customisable at a reasonable cost.

The reason behind the selection of this case study is to show the effectiveness of our method at attempting to maximise value at the requirements level with portfolio thinking.

We will see how the proposed method support satisfying compliance requirements while maximaxing value.To understand the technical details of the SIEM case study, first it is necessary to give a brief explanation of the domain. Figure 6.2 presents an overview of a generic SEIM environment.

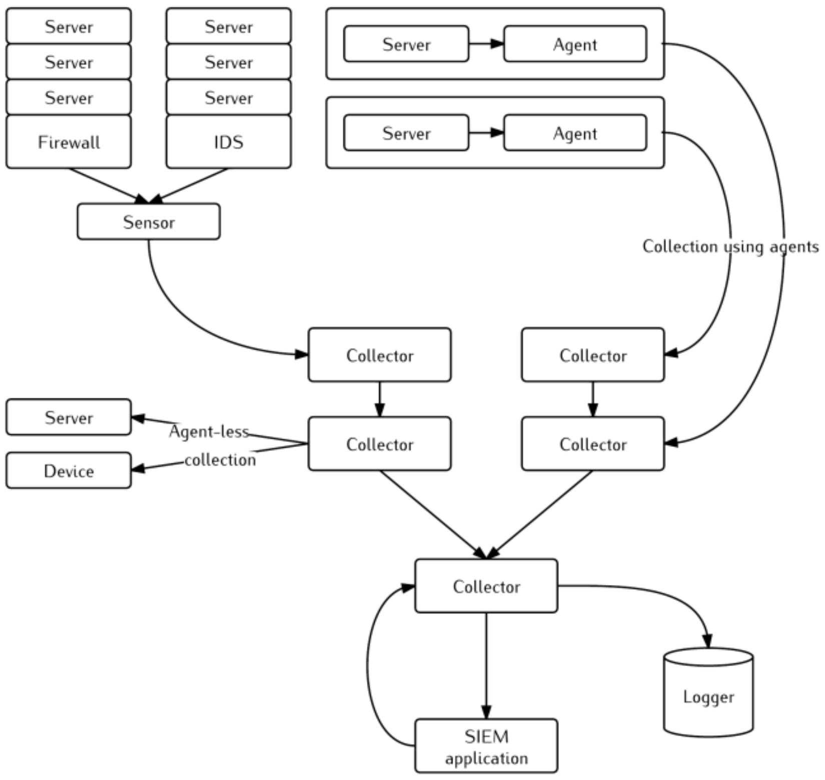


Figure 6.2: An overview of a full SIEM environment

Compliance use cases are based on existing rules and policies contained within information security standards. It can be a valid use case only to focus on such policies. Since SIEM environments are tailored for business rules, such policies only hinder the use case. A SIEM is an extension of log management and is relevant for IT security compliance when it comes to log collection, retention and review.

6.3.1 Rationale

The objective of this case study was to explore the suitability of the method described in Chapter 5 against a real-world system, and to demonstrate the derivation of parameters for an architectural evaluation from the system's compliance requirements. Another objective of this case study is to show how compliance can be built into an application while accounting for its value using the method proposed in this dissertation. The research questions formulated to achieve these objectives are as follows:

1. How to describe, model, and analyse compliance requirements of software systems systematically?

6.3.2 Methodology and Case Study Artefacts

The case study was executed by the PhD candidate, with the cooperation of the staff at the organisation. The PhD candidate played the role of a requirement analyst and an IT security analyst, which was one of the main stakeholders. The steps of the method was simulated the method using the data provided by project.

The case study for ten months. We started following the evaluation process in the early months of the project. We were responsble for the requirement engineering process. This gave us the opportunity to examine the whole process of requirements acquisition, refinement and early implementation. The information used to conduct the case study came from a variety of sources, such as regular meetings with key stakeholders, participation in meetings, and review of available literatures. The case study was conducted by following the phases of the method systematically, as described in Chapter 6. In addition, we assessed the results of the case study against the evaluation criteria presented earlier.

The specific objective of this case study is to investigate how method can be applied by requirement engineers to guide the evaluation of security compliance requirements. The aim is to understand the strengths and weaknesses of the method upon live execution. This case study is presented from the evaluation team's point of view. In the next sections, we follow each one of method's phases in order to evaluate the relevant requirements.

Phase I: Identification We have followed the goal specification phase described in the method to specify the requirements for the system. The first step of the goal specification phase is the elicitation of high-level goals. Table 6.10 shows some high level goals. These high-level goals were further refined into more specific and quantifiable operational goals.

Once lower level goals were sufficiently defined, the evaluation team produced the prequalification goals.

Table 6.10: High Level Goals for the SIEM system

The ability to collect any type of log data regardless of source.
The ability to collect log data with or without installing an agent on the log source device, system or application.
The ability to "normalize" any type of log data for more effective reporting and analysis.
The ability to "scale-down" or "scale-up" dependent upon the environment.
An open architecture allowing direct and secure access to log data via third-party analysis and reporting tools.
A role based security model providing user accountability and access control.
Automated configurable archiving for secure long-term retention of data and events.
Wizard-based retrieval of any archived logs instantly.
The solution shall be capable of performing cross-platform Log Collection
The solution shall be capable of performing log collection for Flat File Logs and all syslog
Agent-less and Agent-based collection.
Log archiving and retrieval.
The ability to alert for specific events from collected logs
Comply with information security requirements

To specify compliance goals, we create a definition of compliance in the context of SIEM applications. This is done by identifying the high-level goals that the compliance programs of the organisation aims to achieve. The compliance to informational security regulations *Compliance [SIEM]* goal is defined as the conjunction of complying with confidentiality, integrity and availability requirements for the data contained within the system. Here, the Integrity [data] goal is refined to Accuracy [data] indicating that integrity in this context refers to the accuracy of the log data contained in system. Integrity may connote additional dimensions not shown in this Figure; such as the need to ensure that only authorized changed can be made to, the accounts and such changes can only be made in an authorized manner. The Confidentiality [data] softgoal connotes that information must be kept private to the system users and against access.

Similarly, the compliance goal is defined in terms of the availability of the SIEM services, denoted by the Availability [Service] goal. This is further refined into Immunity

[Service] connoting the need for application to protect itself from infection by unauthorized undesirable programs.

The goal tree of the system

Figure 6.3 shows the goal tree of the system. The compliance goals are further refined into subgoals. The goals, which point towards the parent goal, are contributing to the satisfaction of the parent goals.

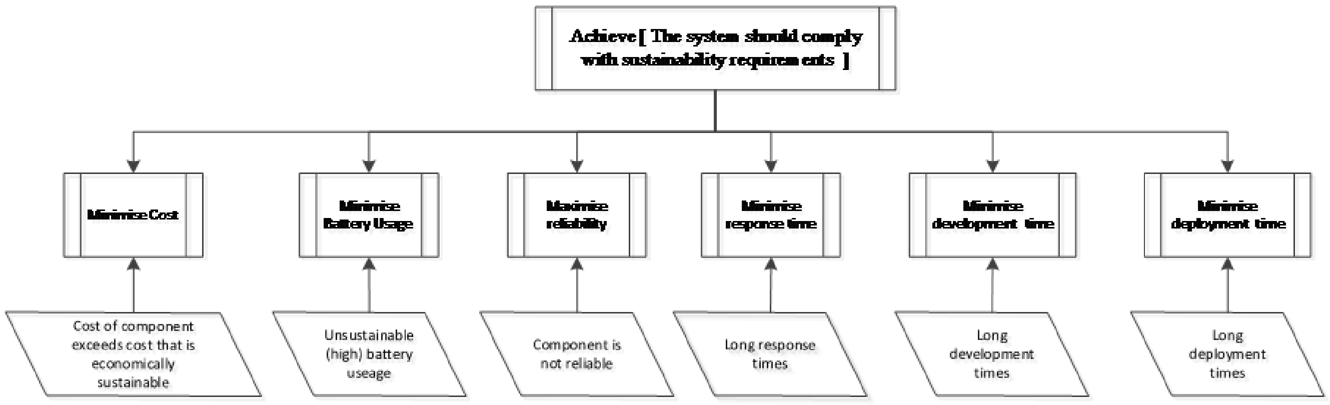

Achieve [The system should comply with sustainability requirements]
Minimise Cost
Minimise Battery Usage
Maximise reliability
Minimise response time
Minimise development time
Minimise deployment time
Cost of component exceeds cost that is economically sustainable
Unsustainable (high) battery useage
Component is not reliable
Long response times
Long development times
Long deployment times

Figure 6.3: Goal Tree of the System

Trace goals to the architecture We have visually modelled what compliance goals needs to be achieved or maintained for the SIEM system. Next, we define *what needs to be done* to achieve the identified goals. These arevisually modelled as operationalizing the goals. For example, Confidentiality is operationalized with the *Encryption [Account]* and *RoleBasedAccessControl[Account]* operationalizing softgoals. In the same way, *Availability[Online Service]* is operationalized with the *Redundancy[Online Service]* softgoal. The goal-based definition of compliance for this application is depicted as discussed above, while the Integrity[Data] goal is refined to a conjunction of Completeness[data] and Accuracy[data] goals. Table 6.11, show a summary of the requirements and the alternative architectural strategies for satisfying the requirements.

The satisficed goals in the Figure shows the snapshot of the design decisions made so far in building compliance into the system.

The main SIEM components, which are needed to operationalise the system requirements of the system are:

- log shipper
- log parser
- log storage, indexing and search
- web interface

Most of the components, depending on the solution, could be replaced by some alternative ones. Multiple functions can be executed by a single part of the log management solution. Single function can be divided among multiple components.

Table 6.11: Summary of Requirements and Compliance Goals

Requirements	**Alternatives**	**Related Compliance Goals**
Protocols	BSD	Authorisation[users]
	Syslog	Authentication[users]
Log Shipper	Graylog	Completeness[data]
	Logstash	Accuracy[data]
	Rsyslog	Scalable[system]
	Syslog-ng	Redundancy[service]
Storage	ElasticSearch	
	Mongo DB	
	MySQL	
Indexing	ElasticSearch	
	Sphinx	
Authentication	Local	
	LDAP	
	None	
Authorisation	Local	
	LDAP	
Log Parser	Grok	
	Json	
	Ruby	
Visualisation interface	Log analyser	
	Kibana	
	Graylog web interface	
	Elsa web interface	

Phase II: Modelling to Assist Valuation

Identity and Quantify Benefits Once the goals have been developed and their values elicited, their relationship to the different information security dimensions is determined. In this study, we use the following information security dimensions:

1. **Confidentiality** [**C**] dimension evaluates the extent to which the AS is likely to contribute and protect confidentiality obligations.
2. **Integrity** [**I**] checks if the goal being evaluated is necessary to protect integrity within the system and its importance in terms of integrity to the system.
3. **Availability** [**A**] criteria checks if the goal being evaluated is necessary to ensure availability of the system.

As mentioned in the previous chapters, we use the following value perspectives to evaluate the impact of an architecture strategy on these dimensions.

1. The financial perspective
2. The customer perspective
3. The internal business process

Since this was an internal project, a system be developed for use for a team also internal to the organisation, only the internal business process perspective was considered. The stakeholders were given the list of 22 architectural strategies that were considered relevant the system to rate. They rated each strategy based on its impact on the identified compliance dimensions. For example, one of the drawbacks of the BSD Syslog protocol is that it uses UDP only. This means there is no delivery control as no acknowledgement of the receipt is made. Another limitation is that BSD syslog does not support encryption, so messages are sent in clear text. It also does not support authentication. In contrast, syslog protocol is a more structured, transmission-reliable and secure than BSD syslog. Hence, syslog was rated higher than BSD as it contributed more towards satisfying the compliance requirements. For simplicity, each they rated each AS by QA (compliance dimensions) on a 3-point scale (1 to 3) based on subjective criteria such as the stakeholder's experience and knowledge. As with CBAM, the stakeholders had to rate each of the compliance dimensions such that the sum of the scores was 100. By design, $\sum$ (DimensionScore) =100 and for all Scores > 1. In this case, based on specific organisational requirements, the scores attributed to the dimensions were Confidentiality = 40, Integrity = 20 and Availability = 40. The benefit for each strategy was calculated using the following formula adapted from CBAM:

$$Benefit = \sum(\ DimnsionScore * Impact)$$

Quantify Cost Implications Quantifying the cost implications was difficult. The costs of the various alternatives were quantified on a scale of high, medium and low, based on the amount of person per month. No guidance regarding cost estimation was given and the developers estimated how much effort to needed to implement the alternative based on their current knowledge.

Quantify Risk Using Obstacle Analysis This requires systematically checking the strategies for obstacles to complying with the compliance goal. Table 6.12 only the estimated criticality of the identified obstacles.

Table 6.12: Compliance Obstacle Criticality

Obstacle	**Criticality**	**Value**
Does not support authentication (RBAC)	High	3
Does not ensure completeness	Medium	2
Does not ensure accuracy	Medium	2
It is not scalable	High	3
No inbuilt redundancy	High	3

In Table 6.13 we compute the risk factor of the achitectural strategy with respect to compliance.

Phase III. Portfolio Analysis After calculating the return and risks associated with each AS, the next step is to determine the overall desirability of a candidate architecture

Table 6.13: Risk Analysis

Requirements	Alternatives	Agent value	obstacle concern	Risk factor
Protocols	BSD	0.875	1	88 %
	Syslog	0.208333	0.25	5%
Log Shipper	Graylog	0.75	1	75%
	Logstash	0.75	1	75%
	Rsyslog	0.625	1	63%
	Syslog-ng	0.75	1	75%
Storage	ElasticSearch	0.625	1	63%
	Mongo DB	0.75	1	75%
	MySQL	0.75	1	75%
Indexing	ElasticSearch	0.208333	0.25	5%
	Sphinx	0.541667	0.625	34%
Authentication	Local	0.541667	0.625	34%
	LDAP	0.208333	0.25	5%
	None	0.875	1	88%
Authorisation	Local	0.541667	0.625	34%
	LDAP	0.208333	0.25	5%
Log Parser	Grok	0.875	1	88%
	Json	0.875	1	88%
	Ruby	0.875	1	88%
Visualisation interface	Kibana	0.375	0.625	23%
	Graylog web interface	0.625	1	63%
	Elsa web interface	0.375	0.625	23%

design, which relates to a portfolio of ASs. The overall effectiveness of an architecture is a function of the total expected return and the global risk of that portfolio.

Following the quantitative analysis, 1298 portfolios were generated. We shortlisted valid architectures that maximise expected net benefits and minimise risks on dimensions related to compliance. An architecture is shortlisted, if it satisfies the relevant constraints such as cost, risk, schedule or the compliance score threshold. One of the compliance requirements elicited was to support authentication. Some of the portfolios became invalid as they didn't support authentication. In this study, the different portfolios were prioritised according to four different (cost, risk, and sharpe ratio) criteria to compare. It was observed that architecture P985 has the highest return and sharpe ratio with a portfolio risk of 34% and a cost of 90 days compared to a portfolio risk of 44% for the cheapest architecture. This architecture was considered optimised for value and risk with respect to the compliance requirements. Table 6.14 shows the components of portfolio P985:

Table 6.14: Portfolio P985

Requirements	Alternatives
Protocols	Syslog
Log Shipper	Rsyslog
Storage	ElasticSearch
Indexing	ElasticSearch
Authentication	LDAP
Log Parser	Json
Visualisation interface	Kibana

The architecture that was eventually decided on by the evaluation team was portfolio P823. This hid a slightly higher risk of 36% and a lower cost of 83 days. Table 6.15 shows the components of portfolio P823:

Table 6.15: Portfolio P823

Requirements	Alternatives
Protocols	Syslog
Log Shipper	Logstash
Storage	ElasticSearch
Indexing	ElasticSearch
Authentication	LDAP
Log Parser	Json
Visualisation interface	Kibana

Phase IV. Technical Debt Analysis We evaluate the architecture by considering the amount of compliance debt that each architecture may incur as the deciding factor.

the portfolio that was choose had a debt of had a relatively low compliance debt of 3%

Using equation 2, we ranked the other architectures from best (low Sd) to worst (high S_D). The portfolio that was chosen had a debt of had a relatively low compliance debt of 3%

Sensitivity Analysis The selection of an architectural portfolio is a multi-criteria decision problem. Benefits, risks, desirability (portfolio value) and debt are important decisions that have not been made or decisions that have been made but whose consequences are not fully understood. The evaluation process can be considered as a decision problem in which parameters have to be estimated and weights are assigned based on human judgments. When solving this type of problem, it is important to study how variations in the model's parameters can affect the result of the analysis and to gain assumptions which are critical. We adopt a one-at-a-time (OAT) approach [154] to perform the parametric sensitivity analysis on the model, to examine the sensitivity of the compliance debt.

To calculate the architecture's sensitivity to change in the return of an individual strategy, we calculate its beta

$$\beta = \frac{Cov\ (x, y)}{Var\ (y)} \tag{1}$$

Where x and y, are unique architectures.

We calculate the sensitivity values for each pair of consecutive alternative architectural portfolios. From the sensitivity analysis, we can infer that an architecture compliance debt's sensitivity to various parameters estimates exhibits the following order of increasing sensitivity

Benefit of AS, obstacle criticality, obstacle likelihood

6.3.3 Results and Discussion

Although method helped to identify what compliance-related information needs to be elicited when engineering requirements, we found the elicitation of quantifiable parameters difficult. While, there were some historical information available, the parameters were estimated by value such as high, medium or low with numerical weights. This was done due to difficulties in eliciting precise quantitative measures and a lack of clear criteria to validate the estimation.

The evaluation process has been valuable to the stakeholders. The basic concept of linking the compliance requirements to the architectural elements was easily understood. Analysing the strategies with respect to compliance was straightforward and it considered the domain where the system will be used. The approach helped to in the selection decision by identifying satisfying portfolios. It also helped to visualise trade-offs using the debt metric. The goal model produced was used to support the development of the system and the analysis was used to support decisions made.

In this study, we encounted some difficulties when the method was applied to a real-world system These limitations are considered as future work in Chapter 7.

1. Modelling uncertainties: The most significant difficulty in the process involved eliciting the values for the model's paramters.

2. Likelihood of compliance obstacles: The likely risk associated with the architectural strategy under development is typically unknown during evaluation. Assessing the likelihood of obstacles affecting each architectural strategy should involve identifying how the obstacles' likelihood varies with how the strategy is implemented. Also potential ways of resolving the obstacles, together with the costs associated with the resoluttion could then be used to decide which architectural strategy should be implemented.

3. Cost: The cost associated with each architectural strategy under consideration was difficult to model in this case study. This was because most of the components were open-sourced. The costs of the various alternatives were quantified on a scale of high, medium and low, based on effort to implement the strategy. This effort was modelled using the amount of person per month.

The method also demonstrated some benefits of the method in a real-world setting.

1. GORE advantages for compliance: GORE was particularly useful for evaluating compliance requirements. The advantages include providing rationale for goals, providing traceability from the goals to the architectural strategies.

2. Obstacle analysis: Obstacle analysis was a systematic way to reason about compliance risks during design-time. The study has demonstrated the application of the compliance obstacle analysis in a real-world setting and offereed the following benefits :

   (a) All expectations associated with each architectural strategy with respect to the compliance requirements were examined and the obstacles were identied

   (b) Accurate modelling of impact for compliance risk: The impact of obstacles of the architectural strategies on the compliance goals can be obtained by the obstacle analysis.

3. Quantitative evaluation of system compliance and stakeholder preferences: The case study has shown that the method can quantitatively evaluate a software system's compliance at design time as well as the satisfaction of the stakeholders with respect to system qualities when system characteristics vary over operational ranges.

4. Value-aware analysis to support decisions: The case study have also demonstrated the suitability of the method for comparing system architectures in a clear and objective way, with emphasis on value and debt.

5. A debt-aware analysis to support decisions

This case study contributed to the research by demonstrating the suitability of our method for a large real-world system. The case study has also highlighted a number of shortcomings on our method.

6.3.4 Threats to Validity

1. This was a one-off study. The results archived from this study do not guarantee that the method and models can be applied to other real-world systems from different domains. This is a threat to the external validity of the study. Due to different constraints, it is not feasible to repeat study in systems from different domains. Stakeholders in our evaluation have formulated compliance requirements and related the

compliance dimensions to steer the evaluation. Consideration of complex/extreme requirements beyond the scope of work presented here is therefore a threat to our results. The method can be applied to any compliance requirement in any domain, as it is not a one-size fits all method. The evaluation framework can be tailored based on different compliance requirements, domain and system characteristics.

2. This study identified, the cost estimation was not estimated by the developers in a metric that could easily fit into the method. This is a threat to the internal validity of the study. In this case, we assumed uniform cost for each strategy and evaluated the architectures using the quantified risk and benefits.

6.4 Critical Evaluation

This thesis focused on providing portfolio-based analysis model and a debt metric for quantitatively evaluating compliance requirements in software systems. A method that brings together the related concepts, models and metric is also described. This chapter has presented two case studies to assess the effectiveness of our method for evaluating compliance requirements. The objective of these case studies is to exemplify the use of our method. The benefits and limitations of each case study have been discussed in its respective section. Due to the strategy, we used to conduct the case studies, we were not able to measure the value added by applying the method. This was because there was no control situation against which to test the results obtained. We acknowledge this is a serious limitation of the evaluation effort.

In this section, we present a critical evaluation of method by assessing the method against the evaluation criteria defined in the beginning of this chapter.

Does the method provide complete, understandable and useful method to guide the evaluation of compliance requirements?

The first aspect we discuss is to what extent the method facilitates the evaluation of compliance requirements. The ease to use is a key aspect that has motivated the development of the method. Several aspects of method contribute to making the method we propose easy to use. For instance, the wide use of guidelines and templates to explain each phase of the method facilitates the application of the method by users without previous experience in evaluating requirements at the architectural level. The method is based on intuitive concepts organized in a structured manner, which contributes to its repeatability. Users of the method do not need to understand the details of the decision theory behind the method. On the other hand, users must have a comprehensive understanding of organization objectives and knowledge about the domain. Given that we have personally conducted the case studies, it is difficult to have a conclusive validation of the overall ease of use of the method. As criticism, we consider that the eliciting the numeric values for the portfolio analysis demanding to apply. The evaluation team preferred to use a qualitative scale ranging from low, medium and high. As a result, it is necessary further evaluation to test the suitability of the prioritisation in large selection projects.

Concerning the completeness of the method, it was observed the method covers all major steps necessary to conduct a disciplined evaluation process. The phases provide a clear, structured process to guide the acquisition and analysis of relevant information. This information assists the evaluation team in making informed decisions, and therefore, choosing an architecture that suits the system and compliance goals. The benefit of method we propose has given to the decision process is that it provides guidance on how to manage risk by quantifying the debt associated with different candidate architectures. The method provides specific mechanisms to address the problem of architectural matching. We evaluate the method using some general qualitative characteristics including simplicity of use, openness, and comprehensiveness

1. A notable desirable feature of method is its flexibility and openness; the method does not define rigorous ways for estimating its parameters, conducting its steps, and confirming specific actions to execute. Consequently, we note that evaluating methods like the method presented in this thesis is rather hard, as their effectiveness is dependent on the way practitioners apply them. In addition, the nature of the

decisions made when applying the fundamentally varies from one project to another, with the addressed problem, and across domains. As a result, the effectiveness of its application is subject to the context in which the model is applied.

2. The method is open; it could be easily integrated to complement existing evaluation methods, with the objective of explicit evaluation for compliance while taking a value-based perspective. The integration may provide a basis for analysing the complexity and economic ramifications of a specific compliance requirements and its impact on the architecture and/or the associated architectural decisions.

Is Goal-based requirements engineering a suitable approach to specify compliance requirements?

Our interest in goal-oriented requirements engineering is based on the fact that goal-oriented requirements engineering starts from the high-level goal which are refined to more concrete goals. This allows traceability links from high-level to low-level goals. Goals are considered more stable [82] and therefore it is reasonable to use goals for requirements specifications. Specifically, GORE contributed to the evaluation by

1. **Ensuring Traceability**: In section 2.4.5, one of the concerns identified for analysing systems for compliance requirements at design-time was traceability, being able to trace choices to the original requirements. Using GORE within the method provided a way for tracing high-level goals to low-level goals and subsequently to the architectural strategies and architectural portfolios. The use of goal-oriented requirements specification was considered to be useful mainly because it allowed the clear separation between strategic objectives and low level requirements, at the same time that it explicitly exhibited the relationship between them in the form of goal tree decomposition

2. **Ensuring Intentionality:** The combined use of GORE for eliciting compliance requirements and the value-driven evaluation was to ensure that the system, once

designed and implemented, will behave with the state described the compliance document.

3. **Ensuring Ability**: In general, any compliance solution is acceptable only if the agent operationalising the compliance requirement, has the capability to perform the necessary actions to achieve the desired goals. By evaluating the architectural strategies benefits and risks with respect to compliance requirements, the method ensured that any portfolio selected had the ability to achieve the desired compliance goals.

Though this thesis has not focused on scenarios related to change and evolution, we argue that the method could be adapted easily to support such evolution. GORE is a sound methodology, which was widely adopted in different application domains. The method has well defined process for refinement, elaboration, specification and operationalization of goals. This method was choose to ease comprehension (due to its wide use the terminologies are defined and we have language to define the goals and obstacles). It is flexible enough to be customized to various solution domains. It explicitly supports the traceability to the decisions made. For all these reasons, we can conclude that goal modelling is an appropriate approach to elicit and specify compliance requirements.

Is the risk analysis strategy using obstacle analysis effective in identifying and analysing risk that may arise during the evaluation process?

Although, in practise, any risk analysis technique can be used with the method, the guidelines presented within the method, modelled risks in goal-oriented form using obstacle analysis. The method helps in identifying and quantifying compliance risks to the architecture. However, we cannot guarantee that the obstacle analysis will be appropriate for risks from other selection contexts. In addition, the correct interpretation and quantification will largely depend on the skills of the evaluation team. These are one of the limitations of our approach.

Does the economics-driven model that exploits portfolio theory provide insights into allocating available resources in a manner, which will minimize

costs, reduce risk and maximize value?

The analogy the method makes with portfolio theory is a simple, yet powerful and comprehensive enough to provide basis for analyses supporting plenty of problems. We have just utilized this simple and intuitive analogy to address two complex compliance-centric design-time problems: valuing the long-term cost-effectiveness of an architecture with respect to compliance requirements and informing the selection based on value. The analogy of the portfolio model with portfolio theory is strength as it is grounded in a sound theory.

Does the model exploit the concept of technical debt to provide a metric for visualising trade-offs between short-term and long-term value in compliance?

Scalability

There are several dimensions for discussing the scalability of the method. Among these dimensions are:

1. Stakeholders Inputs: As the number of stakeholders' input increase, the analysis may become more complex. This is likely to result in more perspectives on value and risk. The situation can become serious in situations where stakeholders are not identified based on relevance and experience. It is important to identify and prioritize the stakeholders involved in the evaluation process. This will limit the number of stakeholders involved in applying the method.

2. Requirements Elicitation: Scalability also becomes an issue if the goal tree becomes too big due to large number of subgoals, we can prune the tree for making the evaluation process easier.

3. Number of architectural strategies: Scalability can also become an issue if architectural strategies to be evaluated are numerous.

4. Portfolio Analysis: The portfolio analysis ensure comparisons of all possible combinations of architectural strategies to form architectural portfolios. This redundancy

makes this process very robust. However, scalability becomes an issue in the portfolio analysis, if this analysis is done manually or with rudimentary tools. Using specialised statistically tools such as R, can make the evaluation faster and easier.

Subjectivity

Most of the selection decisions are based on subjective judgments [7]. The decision to shortlist a specific portfolio is not only based on how well that portfolio is able to meet the technical requirements. Other subjective factors such as monetary constrains may also influence the selection decision. Due to the subjective nature of the goals and the fact that compliance is related to the domain, there may not be clear-cut definition of goal satisfaction. It is therefore important to define a goal satisfaction level [153]. A distinctive feature of the method is the amount of subjectivity used. The method elicits the parameter values from the stakeholders. Since we are using feature analysis method of DESMET [79], which is based on judging methods against some kind of "evaluation criteria" which are subjectively identified. Such evaluations are therefore prone to bias [78]. The assignment of weights due to subjective criteria are approximate. It is therefore important to do the stability analysis of the results obtained from the portfolio analysis. Results are considered stable if order of the solution does not change with slight perturbations in the values of the alternatives. A sensitivity analysis was performed in both case studies to confirm stable results.

6.5 Summary

This chapter described the two case studies used to validate the concepts, models and methods presented in this thesis. The objective of the first case study was to exemplify the use of the method. We have demonstrated how to perform the steps of the method in details. While in the second case study, we have followed a real-real project. The importance of this study is not in the architecture itself, but in how we have used the theory and the model to reason about the value of the architecture in relation to likely compliance requirements. This case study allowed us to establish the suitability of the method to support the selection of complex software systems. Although there are recognised limitations in the method, the principal benefit of method is the systematic support for identifying and evaluating compliance requirements for value at the requirements and architectural level.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Stay hungry, stay foolish.

Steve Jobs

This thesis has researched the use of value-based approaches grounded on the principle of portfolio theory to evaluate compliance requirements at design time. The goal of the work presented in this thesis has been the development of new methods to improve the quality of decisions made during the evaluation of compliance at design-time. In this chapter, we summarize the thesis contribution. We highlight some future work.

7.1 Summary of Contribution

The main goal of the thesis has been the development of a framework for systematically evaluating the compliance requirements at the architectural level, taking a value-driven approach.

We have provided background on the related concepts and have reviewed work on compliance requirements. We have explored the relationship between compliance goals and other goals such as system, quality and business goals. We have investigated the criteria for evaluating compliance from a value-driven perspective and have described a method to address these requirements.

This thesis contributes to value-based evaluation of compliance requirements. More specifically, the work presented in this thesis employs a goal-oriented approach with a portfolio model and a compliance debt based on the concept of technical debt to provide a value aware approach to compliance. This thesis brings together the related concepts, the portfolio model, compliance debt metric and the methods. It provides guidelines on eliciting compliance requirements and relating the relevant architectural decisions to value. We have demonstrated the valuation points of view in capturing the value from different compliance dimensions and value perspectives. This framework can inform the selection decisions relating to compliance requirements in the context of architecture, hoping to maximise the value-creation with respect to some valuation points of view (such as costs and risks). This method can be applied at the early phases of the development lifecycle to encourages a proactive approach to compliance in system.

The thesis makes the following specific contributions:

1. This thesis has contributed a novel economic- driven model that exploits portfolio theory to compliance requirements at the architectural level based on portfolio theory. The model provides insights into allocating available resources in a manner, which maximizes value. The modification to the general portfolio optimisation formulation explicitly integrates aspects of portfolio valuation with a focus on evaluating compliance requirements in architectures. In the evaluation, costs and risks are associated with architectural strategies for satisfying compliance requirements. To capitalize on the architectural portfolio, requirements and architectural strategies have to be viewed as the pieces of a puzzle for creating value for the stakeholders. By having this broader view, it also becomes easier to find strategies that can be combined in order to reduce the risk since the entire architecture will be taken into account.

2. A novel approach that devises a model that exploits the concept of technical debt as a metric for visualising trade-offs between short-term and long-term value in compliance. This thesis introduced a new dimension of technical debt for explicating compliance debt at the architecture-level. This type of debt is a strategic debt that can be attributed to the gaps between an ideal architecture satisfying all compliance

requirements completely and the selected architectural portfolio. The debt can be attributed to different situation such as poor architecting, budget restrictions, not satisfying the requirements and poor decision-making. This approach models compliance debt as a function of compliance requirements tradeoffs in the architecture. It recommends the desirable cost-effective solutions for given time of interest for specific requirements

3. A systematic five-phase method, which combines goal oriented requirements engineering, the portfolio model and technical debt model for addressing compliance at design-time. By evaluating the architectural portfolio, it is possible to evaluate and compare quantitatively how complaint the architecture is. The method presented in this thesis defines a set of steps that describes possible ways for estimating the model parameters, applying the models and performing the analysis. The method is a systematic and iterative method structured into five phases. Each phase of the method presents a number of techniques and guidelines to assist in the analysis of compliance requirements.

4. The thesis has also contributed to literature search and documentation that is relates investigation - compliance, value-based software architecture, technical debt among others

5. In evaluating the thesis, we have explored the approach suitability in addressing the research question. Two case studies demonstrated the applicability of the methods described in this thesis. We have used general qualitative characteristics including simplicity of use, openness, comprehensiveness, scalability and subjectivity to evaluate the applicability of the method further.

7.2 Lessons Learned

We now summarise the lessons learned:

1. The evaluation method largely depends on the expertise of the analysts and developers. From conducting the second case study, a number of benefits of having a disciplined method to guide the evaluation where observed. By following the phases presented in method, it was possible to provide a systematic and repeatable process of gathering relevant information, analysing alternatives and making decisions. However, it was also recognize that no matter how objective, there are still underlying values and subjectiveness in the ways that people perceive, judge and ultimately make decisions. Consequently, we consider that largely the success o fthe selection process depends on the experience and domain knowledge of the evaluation team.

2. The method to guide the evaluation needs to be customisable. The method provided useful guidance to conduct the case studies. Some parts of the method offered valuable support, such evaluating alternative architectures for value and debt. On the other hand, approaches such as the obstacle analysis technique, although it was beneficial to the project, their use was considered complex and time demanding. Even when using a different risk analysis approach, the evaluation exercise proved to be valid. In addition to that, it is obvious that there is no one fits all solution to developing software systems.

3. Good understanding of organizational requirements is vital for the success of selection process

4. Compliance Debt is subjective. Compliance debt can be affected by the following: (1) The decision makers and their expertise (e.g. stakeholders), (2) The elicited parameter values, (3) the importance of the compliance requirements to the system design and (4) the potential use and value of the system. The threshold of compliance debt depends on the acceptance of stakeholders.

5. Value is subjective. Value depends on the different variables. This can be modelled as a function of the utility of the system, the dependencies, the acceptance-level of the stakeholders

7.3 Future Work

The work presented in this thesis can be improved and extended in several directions. In this section, we explore immediate future work for building on this thesis.

7.3.1 Multi-objective optimization view to evaluation and the interdependence of compliance and non-functional requirements

We have taken the view that software design and engineering activity is one of investing resources with the aim of maximizing the value [171]. It is possible to adopt a complex view of value. One could characterize engineering compliance requirements as a multi-objective optimization activity in which one trades performance for compliance, or in which one satisfies multiple stakeholders. We have taken a narrow view to valuation: the value is measured relative to one objective (compliance) at a time. Although we treated compliance was a concern, which affects other non-functional requirements, we have not considered the effect of ensuring compliance on other behavioral requirements like scalability and reliability. Evaluating this impact could make the analysis complex. Compliance requirements could affect other non-functional requirements positively or negatively. An important point for complying with information security requirements is to ensure confidentiality of the system. In this case, the compliance requirement affects security requirements positively. The framework could be then complemented by means for measuring the additional cost and risk (cost savings and risk mitigation) due to the impact of compliance requirements on other non-functional requirements.

7.3.2 Viewpoints Valuation for Compliance Requirements

In today's world of rapidly changing technologies and exploits, compliance requirements need to be revisited to better cater for these changes. This necessitates finding a comprehensive solution for capturing the value from different perspectives. In chapter 5, we have highlighted a solution for addressing this problem. Using value perspectives and compliance dimensions aims at providing a solution for quantifying the value from viewpoints. Future work may entail finding ways to manage the valuation such as identifying the dimensions, which are critical for understanding the compliance requirements, prioritizing the valuation of these dimensions and value perspectives and managing trade-offs. The method's interpretations and decision-making will need to be modified accordingly. The accuracy of the method can be improved with a more comprehensive approach to capturing value from different perspectives. This has a promise of accommodating diversity through the identification perspectives and concerns.

7.3.3 Tangible and Intangible Debt in Compliance

We have only examined one aspect of debt as defined in Chapter 4. In this perspective, we have a narrow and limited view of debt which relates to how much we lag behind the optimal architecture. Future work may look at how we can extend our understanding for debt in relation to compliance to cover both the tangible and intangible consequences of the debt. This will require rethinking how to compute the principal and the interest of intangible debt (for example debt that is not visible). This calls for qualitative approaches for eliciting , defining , negotiating and reconciling values, that relates to principal and interest. We are currently investigating the extending the use of GQM [23] for reasoning about intangible debt.

7.3.4 Compliance at Run-time

The thesis has looked at compliance requirements at design-time. We are interested in how the framework can be modified and used to evaluate compliance at run-time.Compliance at run-time aims at ensuring compliance when the system has been designed and in use. The analysis usually involves monitoring of the system at run-time to ensure adherence to compliance requirement specifications. Run-time compliance evaluation is retrospective and can be used for empirically evaluating a system for compliance to calibrate the design-time evaluation results.

APPENDIX A

PUBLICATIONS

The work presented in this thesis is based on and extends the followings papers that have been published during the course of the PhD programme.

1. ”Systematic elaboration of compliance requirements using compliance debt and portfolio theory.” Ojameruaye Bendra, and Rami Bahsoon. Requirements Engineering: Foundation for Software Quality, 2014

2. ”A risk-aware framework for compliance goal-obstacle analysis.” Ojameruaye Bendra, and Rami Bahsoon. Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM, 2015.

3. ”Sustainability Debt: A portfolio-based approach for evaluating sustainability requirements in architectures”. Ojameruaye Bendra, Rami Bahsoon and Leticia Duboc. International Conference on Software Engineering ICSE, 2016

APPENDIX B

JAVA CODE FOR IMPLEMENTATION

```
SUSEvaluation Class
================================
package evaluation.model;

import java.util.ArrayList;
import java.util.List;

import excelReadWrite.ReadExcelClass;
import excelReadWrite.WriteToExcel;

public class SUSEvaluation {
private static String sus_evaluationInputExcelPath = "C:\\
    sus_evaluation.xlsx";
private static String sus_evaluationResultExcelPath = "C:\\
    sus_evaluation_result.xlsx";

/**
* Loads Possible Decisions from Excel File
* @param dataStartRow - This specifies the starting row of input
```

```
    data in the input excel file
* @return - Returns List of loaded Possible Decisions.
*/
private static List<PossibleDecision> getPossibleDecisions(int
    dataStartRow){
String[][] possibleDecisionsArry;
possibleDecisionsArry = ReadExcelClass
.excelread(sus_evaluationInputExcelPath);

PossibleDecision pd = new PossibleDecision();
pd.setName("");

List<PossibleDecision> createdPossibleDecisions = new ArrayList<
    PossibleDecision>();


for (int i = (dataStartRow - 1); i < possibleDecisionsArry.
    length; i++) {
//[0][0] = Row 1 Column 1 and [0][1] = Row 1 Column 2

//Checking if Decision has value in the merged cell
if(!possibleDecisionsArry[i][0].equals("false") && !
    possibleDecisionsArry[i][0].equals(pd.getName())){
//new Possible Decision Met
//save old possible Decision

pd = new PossibleDecision();

createdPossibleDecisions.add(pd);

pd.setName(possibleDecisionsArry[i][0]);

pd.setWeight(Double.valueOf(possibleDecisionsArry[i][1]));

pd.addAlternative(loadAlternative(possibleDecisionsArry, i));

}else{
pd.addAlternative(loadAlternative(possibleDecisionsArry, i));
}

}

return createdPossibleDecisions;
}
/**
* creates an alternative object from the row in excel file
* @param possibleDecisiondataArry - Contains all data in input
    excel
* @param row - Current row from which to find alternative
    details
* @return
*/
private static Alternative loadAlternative(String[][]
    possibleDecisiondataArry, int row){
Alternative alt = new Alternative();
alt.setName(possibleDecisiondataArry[row][2]);
alt.setCost(Double.valueOf(possibleDecisiondataArry[row][3]));
alt.setAreturn(Double.valueOf(possibleDecisiondataArry[row][4]))
    ;
alt.setRisk(Double.valueOf(possibleDecisiondataArry[row][5]));

return alt;
}
/**
* Computes sus_evaluation results and also writes the result to
    excel
* @param args
*/
public static void main(String[] args){
List<PossibleDecision> possibledecisions = getPossibleDecisions
    (2);
List<Decision> temp_portfolioDecisions = new ArrayList<Decision
    >();
List<Portfolio> generatedPortfolios = new ArrayList<Portfolio>()
    ;

int possibleDecisionIndex =0;

computePortfolios(temp_portfolioDecisions,generatedPortfolios
,possibleDecisionIndex,possibledecisions);

//Write generated Portfolios to Excel
WriteToExcel.WriteToExcelFile(generatedPortfolios,
    sus_evaluationResultExcelPath);
```

```

}
/**
* Recursively Generates Portfolios and Decisions used in
    creating the portfolios
* Method starts from the given possible decision index, it then
    generates all possible decisions for
* that possible decision object. But for each decision in the
    PossibleDecisions Objects, a check is made
* if there are any other possible decision. This allows for one
    decision to be selected from the next
* possible decisions object.
* Example for below possible decisions
* A {B,C}
* D {E, F, K}
* G {H,I}
* Recursion generates 12 portfolios as below using calculations
    from input 2^2 x 3^1 = 12
* {AB, DE, GH}
* {AB, DE, GI}
* {AB, DF, GH}
* {AB, DF, GI}
* .
* .
* .
* {AC, DK, GI}
* @param currentPortfolioDecisions - Decisions from each recurse
    , a new portfolio is generated using this at
```

```
* the last recurse
* @param pf - List of generated portfolios from the method
* @param possibleDecisionIndex - index of current or starting
    possible decisions
* @param possibledecisions - List of possible decisions with all
     alternatives
*/
public static void computePortfolios(List<Decision>
    currentPortfolioDecisions, List<Portfolio> pf, int
    possibleDecisionIndex
, List<PossibleDecision> possibledecisions ){

//Recurse if there are further possible Decisions, recurse
boolean isToRecurse = (possibledecisions.size() - 1) >
    possibleDecisionIndex;

//Take a copy of received decision and use it for subsequent
    decisions
List<Decision> receivedPortfolioDecisions =
    currentPortfolioDecisions;

//create a temporary generated portfolio decisions and populate
    it with next decision
List<Decision> tempPortfolioDecisions = new ArrayList<Decision>(
    currentPortfolioDecisions);

//Take a copy of the possible decision index to remember current
     possible decision
```

```
int tempCnt = possibleDecisionIndex + 0;

PossibleDecision pd = possibledecisions.get(
    possibleDecisionIndex);
//Create all possible decisions in this Possible decision
for(int a=0 ; a < pd.getAlternatives().size(); a++){

Decision d = new Decision();
d.setName(pd.getName());
d.setWeight(pd.getWeight());
d.setAlternative(pd.getAlternatives().get(a));

tempPortfolioDecisions.add(d);

if(isToRecurse){
//There are more possible decisions, recurse
possibleDecisionIndex++;
computePortfolios(tempPortfolioDecisions,pf,
    possibleDecisionIndex,possibledecisions);
}else{
//This possible decision is the last possible decision in the
    list hence this alternative
// is the last, portfolio can now be created using the list of
    decisions
Portfolio p = new Portfolio(tempPortfolioDecisions);
pf.add(p);
}
//retrieve current possible decision
possibleDecisionIndex = tempCnt + 0;
//Initialize a new temporary list of the next portfolio's
    decision
tempPortfolioDecisions = new ArrayList<Decision>(
    currentPortfolioDecisions);
}

}

}
==========================
Possible Decision Class
==========================
package evaluation.model;

import java.util.ArrayList;
import java.util.List;

public class PossibleDecision {
private String name;
private double weight;
private List<Alternative> alternatives;

public PossibleDecision(){
alternatives = new ArrayList<Alternative>();
}

public void addAlternative(Alternative alternative){
```

```
alternatives.add(alternative);
}
/**
* @return the name
*/
public String getName() {
return name;
}
/**
* @param name the name to set
*/
public void setName(String name) {
this.name = name;
}
/**
* @return the weight
*/
public double getWeight() {
return weight;
}
/**
* @param weight the weight to set
*/
public void setWeight(double weight) {
this.weight = weight;
}
/**
* @return the alternatives
```

```
*/
public List<Alternative> getAlternatives() {
return alternatives;
}
/**
* @param alternatives the alternatives to set
*/
public void setAlternatives(List<Alternative> alternatives) {
this.alternatives = alternatives;
}
}
==========================
Portfolio Class
==========================
package evaluation.model;

import java.util.List;

public class Portfolio {
private List<Decision> decisions;
private double totalcost;
private double expectedreturn;
private double expectedrisk;

public Portfolio(List<Decision> decisions){
this.decisions = decisions;

for (Decision decs : decisions){
```

```
totalcost+=decs.getAlternative().getCost();
expectedreturn+=(decs.getAlternative().getAreturn() * decs.
    getWeight());
expectedrisk+=(decs.getAlternative().getRisk() * decs.getWeight
    ());
}

}

/**
* @return the decisions
*/
public List<Decision> getDecisions() {
return decisions;
}

/**
* @param decisions the decisions to set
*/
public void setDecisions(List<Decision> decisions) {
this.decisions = decisions;
}

/**
* @return the totalcost
*/
public double getTotalcost() {
return totalcost;
```

```
}

/**
* @param totalcost the totalcost to set
*/
public void setTotalcost(double totalcost) {
this.totalcost = totalcost;
}

/**
* @return the expectedreturn
*/
public double getExpectedreturn() {
return expectedreturn;
}

/**
* @param expectedreturn the expectedreturn to set
*/
public void setExpectedreturn(double expectedreturn) {
this.expectedreturn = expectedreturn;
}

/**
* @return the expectedrisk
*/
public double getExpectedrisk() {
return expectedrisk;
```

```
}

/**
* @param expectedrisk the expectedrisk to set
*/
public void setExpectedrisk(double expectedrisk) {
this.expectedrisk = expectedrisk;
}

}
==========================
Decision Class
==========================
package evaluation.model;


public class Decision {
private String name;
private double weight;
private Alternative alternative;
/**
* @return the name
*/
public String getName() {
return name;
}
/**
* @param name the name to set
```

```
*/
public void setName(String name) {
this.name = name;
}
/**
* @return the weight
*/
public double getWeight() {
return weight;
}
/**
* @param weight the weight to set
*/
public void setWeight(double weight) {
this.weight = weight;
}
/**
* @return the alternative
*/
public Alternative getAlternative() {
return alternative;
}
/**
* @param alternative the alternative to set
*/
public void setAlternative(Alternative alternative) {
this.alternative = alternative;
}
```

```

}
==============================
Alternative Class
==============================
package evaluation.model;

public class Alternative {
private double cost;
private double areturn;
private double risk;
private String name;

/**
* @return the cost
*/
public double getCost() {
return cost;
}
/**
* @param cost the cost to set
*/
public void setCost(double cost) {
this.cost = cost;
}
/**
* @return the areturn
*/
```

```java
public double getAreturn() {
return areturn;
}
/**
* @param areturn the areturn to set
*/
public void setAreturn(double areturn) {
this.areturn = areturn;
}
/**
* @return the risk
*/
public double getRisk() {
return risk;
}
/**
* @param risk the risk to set
*/
public void setRisk(double risk) {
this.risk = risk;
}
/**
* @return the name
*/
public String getName() {
return name;
}
/**
```

```
* @param name the name to set
*/
public void setName(String name) {
this.name = name;
}

}
=============================
ReadExcelClass Class
=============================
package excelReadWrite;

import java.io.File;
import java.io.FileInputStream;

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

//Class for reading the excel test scripts
public class ReadExcelClass {

//Method for reading the excel test scripts
public static String[][] excelread(String excelPath) {

//data array to store excel data
```

```
String[][] data = null;
try {
// Retrieving the excel test script based on the path
File excel = new File(excelPath);

FileInputStream fis = new FileInputStream(excel);

XSSFWorkbook wb = new XSSFWorkbook(fis);
XSSFSheet ws = wb.getSheet("input");

int rowNum = ws.getLastRowNum() + 1;
int colNum = ws.getRow(0).getLastCellNum();
data = new String[rowNum][colNum];

// Loop to read cell values and store it in the data array
for (int i = 0; i < rowNum; i++) {

XSSFRow row = ws.getRow(i);

for (int j = 0; j < colNum; j++) {

XSSFCell cell = row.getCell(j);
String value = cellToString(cell);
data[i][j] = value;

//System.out.println("The value is:" + value);

}
}
} catch (Exception ex) {
ex.printStackTrace();
}
return data;
}

//method for converting cell values to string
public static String cellToString(XSSFCell cell) {

int type;
Object result;
type = cell.getCellType();


switch (type) {

case Cell.CELL_TYPE_NUMERIC: // numeric value in Excel
case Cell.CELL_TYPE_FORMULA: // precomputed value based on
   formula
result = cell.getNumericCellValue();
break;
case Cell.CELL_TYPE_STRING: // String Value in Excel
result = cell.getStringCellValue();
break;
case Cell.CELL_TYPE_BLANK:
result = "";
case Cell.CELL_TYPE_BOOLEAN: //boolean value
```

```
result =  cell.getBooleanCellValue();
break;
case Cell.CELL_TYPE_ERROR:
default:
throw new RuntimeException("There is no support for this type of
    cell");
}


return result.toString();

}
}
================================
WriteToExcel Class
================================
package excelReadWrite;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.poi.ss.usermodel.WorkbookFactory;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import evaluation.model.Decision;
import evaluation.model.Portfolio;

public class WriteToExcel {


public static void WriteToExcelFile(List<Portfolio> portfolios,
    String WritePath){

try {
// Retrieving the excel test script based on the path
File excel = new File(WritePath);

//FileInputStream fis = new FileInputStream(excel);
FileOutputStream fos=new FileOutputStream(excel);

XSSFWorkbook wb = new XSSFWorkbook();

XSSFSheet ws = wb.createSheet("Result");


int rowNum =0;

XSSFRow row = ws.createRow(rowNum++);
```

```java
XSSFCell cell = row.createCell(0);
cell.setCellValue("Decision");

//Populate Column Header
int colNum = 0;
for(Decision ds : portfolios.get(0).getDecisions()){
cell = row.createCell(++colNum);
cell.setCellValue(ds.getName());
}
colNum = 0;

//rowNum= populatePortFolioRows(rowNum,portfolios.get(0),ws,1);
//rowNum= populatePortFolioRows(rowNum,portfolios.get(1),ws,2);
Iterator<Portfolio> iterator = portfolios.iterator();
int pf_indx = 1;
while(iterator.hasNext()){
Portfolio pf =iterator.next();
rowNum = populatePortFolioRows(rowNum,pf,ws,pf_indx++);
//System.out.println("Excisting numbers"+part);
//row = ws.createRow(rowNum++);
//XSSFCell cell = row.createCell(0);
//cell.setCellValue(part);

}

wb.write(fos);
fos.close();
System.out.println(" written successfully");
```

```
} catch (Exception ex) {
ex.printStackTrace();
}
}

public static int populatePortFolioRows(int lastRowNum,
    Portfolio portfolio , XSSFSheet ws, int portfolioIndx){

//Populate Column Header
int colNum = 0;

XSSFRow rowAltNames = ws.createRow(++lastRowNum);
XSSFRow rowCost = ws.createRow(++lastRowNum);
XSSFRow rowReturn = ws.createRow(++lastRowNum);
XSSFRow rowRisk = ws.createRow(++lastRowNum);

rowAltNames.createCell(0).setCellValue("P" + portfolioIndx);
rowCost.createCell(0).setCellValue("Costs");
rowReturn.createCell(0).setCellValue("Return");
rowRisk.createCell(0).setCellValue("Risk");
//Populate the Alternatives
int tmp_colNum = ++colNum;
for(Decision ds : portfolio.getDecisions()){

rowAltNames.createCell(tmp_colNum).setCellValue(ds.
    getAlternative().getName());
rowCost.createCell(tmp_colNum).setCellValue((new DecimalFormat("
    #0.00").format(ds.getAlternative().getCost())));
```

```
rowReturn.createCell(tmp_colNum).setCellValue((new DecimalFormat
    ("#0.00").format(ds.getAlternative().getAreturn())));
rowRisk.createCell(tmp_colNum).setCellValue((new DecimalFormat("
    #0.00").format(ds.getAlternative().getRisk())));
tmp_colNum = ++colNum;

}
//Populate Total Cost, EXpected Risk, Expected Return
rowAltNames.createCell(tmp_colNum).setCellValue("Total Cost");
rowCost.createCell(tmp_colNum).setCellValue((new DecimalFormat("
    #0.00").format(portfolio.getTotalcost())));
tmp_colNum = ++colNum;
rowAltNames.createCell(tmp_colNum).setCellValue("Expected Return
    ");
rowCost.createCell(tmp_colNum).setCellValue((new DecimalFormat("
    #0.00").format(portfolio.getExpectedreturn())));
tmp_colNum = ++colNum;
rowAltNames.createCell(tmp_colNum).setCellValue("Expected Risk")
    ;
rowCost.createCell(tmp_colNum).setCellValue((new DecimalFormat("
    #0.00").format(portfolio.getExpectedrisk())));

return lastRowNum;

}
}
==================================
Libraries
```

```
==============================
dom4j-1.6.1.jar          poi-examples-3.9.jar          poi-scratchpad
    -3.9.jar
poi-3.9.jar              poi-ooxml-3.9.jar             xmlbeans-2.3.0.
    jar
poi-contrib-3.6.jar      poi-ooxml-schemas-3.9.jar
```

LIST OF REFERENCES

[1] Norris Syed Abdullah, Shazia Sadiq, and Marta Indulska. Emerging challenges in information systems research for regulatory compliance management. In *Advanced information systems engineering*, pages 251–265. Springer, 2010. One citation in section 2.4.

[2] Carina Alves and Anthony Finkelsteiin. Challenges in cots decision-making: A goal-driven requirements engineering perspective. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE '02, pages 789–794, New York, NY, USA, 2002. ACM. One citation in section 2.2.

[3] Carina Frota Alves. Managing Mismatches in COTS-Based Development. (September), 2005. 3 citations in sections 2.2, 5.3.1, and 5.3.1.

[4] Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes, and Rodrigo O Spinola. Towards an ontology of terms on technical debt. In *Managing Technical Debt (MTD), 2014 Sixth International Workshop on*, pages 1–7. IEEE, 2014. 3 citations in sections 2, 6, and 8.

[5] Esra Alzaghoul and Rami Bahsoon. CloudMTD: Using real options to manage technical debt in cloud-based service selection. *2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings*, pages 55–62, 2013. One citation in section 1.

[6] Esra Alzaghoul and Rami Bahsoon. Economics-driven approach for managing technical debt in cloud-based architectures. *Proceedings - 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013*, pages 239–242, 2013. 2 citations in sections 1.1 and 11.

[7] A.I. Anton. Goal-based requirements analysis. In *Proceedings of the Second International Conference on Requirements Engineering*, pages 136–144. IEEE Comput. Soc. Press, 1996. One citation in section 5.1.

[8] Annie I. Antón, Ryan a. Carter, Aldo Dagnino, John H. Dempster, and Devon F. Siege. Deriving Goals from a Use-Case Based Requirements Specification. *Requirements Engineering*, 6(1):63–73, 2001. One citation in section 2.2.

[9] S Anwer and N Ikram. Goal Oriented Requirement Engineering: A Critical Study of Techniques. *2006 13th Asia Pacific Software Engineering Conference APSEC06*, pages 121–130, 2006. One citation in section 2.2.

[10] M Arnesano, A P Carlucci, and D Laforgia. Extension of portfolio theory application to energy planning problem âĂŞ The Italian case. *Sustainable Energy and Environmental Protection 2010*, 39(1):112–124, 2012. One citation in section 3.1.2.

[11] Naveen Ashish, Ronald Eguchi, Rajesh Hegde, Charles Huyck, Dmitri Kalashnikov, Sharad Mehrotra, Padhraic Smyth, and Nalini Venkatasubramanian. Situational Awareness Technologies for Disaster Response. *Terrorism Informatics: Knowledge Management and Data Mining for Homeland Security*, pages 517–544, 2008. One citation in section 6.2.1.

[12] Yudistira Asnar, Paolo Giorgini, and John Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, 16(2):101–116, sep 2011. One citation in section 2.2.1.

[13] Yudistira Asnar and Fabio Massacci. A method for security governance, risk, and compliance (GRC): A goal-process approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6858 LNCS:152–184, 2011. One citation in section 2.5.

[14] S Awerbuch. Portfolio-Based Electricity Generation Planning: Policy Implications for Renewables and . . . . *Mitigation and Adaptation Strategies for Global Change*, 2006. One citation in section 3.1.2.

[15] S Awerbuch and M Berger. Applying portfolio theory to EU electricity planning and policy-making. *IEA/EET working paper*, 3(February):69, 2003. One citation in section 3.1.2.

[16] Shimon Awerbuch and Spencer Yang. Using Portfolio Theory to Value Power Generation Investments. *Analytical Methods for Energy Diversity and Security*, (2007):61–68, 2008. One citation in section 3.1.2.

[17] John Bace, Carol Rozwell, Joseph Feiman, and Bill Kirwin. Gartner Research: Understanding the costs of compliance. *Gartner Research*, (July):1–19, 2006. One citation in section 2.4.

[18] Rami Bahsoon and Wolfgang Emmerich. ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architecture. *Proceedings of the Fifth Workshop on Economics-Driven Software Engineering Research, EDSER 5, held in conjunction with the 25 th International Conference on Software Engineering*, pages 1–10, 2003. One citation in section 4.

[19] Rami Bahsoon and Wolfgang Emmerich. Requirements for evaluating architectural stability. In *IEEE International Conference on Computer Systems and Applications, 2006*, volume 2006, pages 1143–1146, 2006. One citation in section 2.3.2.

[20] I Bardhan, R Kauffman, and S Naranpanawe. IT project portfolio optimization: A risk management approach to software development governance. *IBM Journal of Research and Development*, 54(2):2:1–2:18, 2010. One citation in section 3.1.2.

[21] Sebastian Barney, Ayb??ke Aurum, and Claes Wohlin. A product management challenge: Creating software product value through requirements selection. *Journal of Systems Architecture*, 54(6):576–593, 2008. 2 citations in sections 1 and 2.3.1.

[22] Brent Barton and Chris Sterling. Manage project portfolios more effectively by including software debt in the decision process. *Cutter IT Journal*, 23(10):19, 2010. One citation in section 4.3.

[23] Victor Basili, Gianluigi Caldiera, HD Rombach, and VRBG Caldiera. The goal question metric approach. *Encyclopedia of . . .*, 2:1–10, 1994. 2 citations in sections 5.4.1 and 7.3.3.

[24] Christoph Becker, Ruzanna Chitchyan, Leticia Duboc, Steve Easterbrook, Birgit Penzenstadler, Norbert Seyff, Colin C Venters, Ruzanna Chitchyan, Norbert Seyff, Leticia Duboc, Colin C Venters, and Steve Easterbrook. Sustainability Design and Software: The Karlskrona Manifesto. *Proceedings - International Conference on Software Engineering*, 2:467–476, 2015. 2 citations in sections 6.2.1 and 6.2.2.

[25] Rami Bahsoon Bendra Ojameruaye. Systematic Elaboration of Compliance Requirements Using Compliance Debt and Portfolio Theory. *Requirements Engineering: Foundation for Software Quality*, 8396:152–167, 2014. 7 citations in sections 2.2.1, 3.1.1, 3.1.2, 3.2.1, 1, 5, and 5.4.3.

[26] Stefanie Betz, Christoph Becker, Ruzanna Chitchyan, Leticia Duboc, Steve M. Easterbrook, Birgit Penzenstadler, Norbert Seyff, and Colin C. Venters. Sustainability debt: A metaphor to support Sustainability design decisions. *CEUR Workshop Proceedings*, 1416:55–63, 2015. One citation in section 6.2.1.

[27] Bodil Stilling Blichfeldt and Pernille Eskerod. Project portfolio management - There's more to it than what management enacts. *International Journal of Project Management*, 26(4):357–365, 2008. One citation in section 3.1.2.

[28] B W Boehm and T DeMarco. Software risk management. *IEEE Software*, 14(3):17–19, 1997. One citation in section 3.2.3.

[29] Barry Boehm. Software risk management: principles and practices. *IEEE Software*, 8(January):32–41, 1991. One citation in section 3.2.3.

[30] Barry Boehm. Value-based software engineering. *ACM SIGSOFT Software Engineering Notes*, 28(2):3–, mar 2003. 6 citations in sections 1.1, 1.4, 2.3, 2.3, 2.3.2, and 2.3.3.

[31] Barry Boehm, Alexander Egyed, Julie Kwan, Dan Port, Archita Shah, and Ray Madachy. Using the WinWin spiral model: A case study. *Computer*, 31(7):33–44, 1998. One citation in section 2.3.

[32] Barry Boehm and Li Guo Huang. Value-based software engineering: A case study. *Computer*, 36(3):33–41+4, 2003. 4 citations in sections 2.3, 2.6, 3.2.1, and 3.2.3.

[33] Barry W. Boehm and Apurva Jain. An initial theory of value-based software engineering. In *Value-Based Software Engineering*, pages 15–37. Springer Berlin Heidelberg, 2006. 2 citations in sections (document) and 2.1.

[34] BW Boehm. Value-based software engineering: Overview and agenda. *Value-Based Software Engineering*, (February):1–16, 2006. One citation in section 2.3.

[35] BW W Barry Boehm, KJ Kevin KJ J Sullivan, and Thornton Hall. Software economics: a roadmap. *... of the conference on The future of Software ...*, pages 319–343, 2000. One citation in section 3.2.1.

[36] Johannes Bohnet and Jürgen Döllner. Monitoring code quality and development activity by software maps. *Proceeding of the 2nd working on Managing technical debt - MTD '11*, page 9, 2011. 2 citations in sections 3 and 2.

[37] Jean-Philippe Bouchaud and Marc Potters. *Theory of financial risk and derivative pricing: from statistical physics to risk management*. Cambridge university press, 2003. One citation in section 3.4.

[38] TD Travis Durand Breaux. *Legal requirements acquisition for the specification of legally compliant information systems*. 2009. 2 citations in sections 1 and 2.

[39] Travis D. Breaux, Annie AntÃÂşn, Eugene H. Spafford, and Annie I. Antón. A distributed requirements management framework for legal compliance and accountability. *Computers & Security*, 28(1-2):8–17, feb 2009. One citation in section 2.5.

[40] Travis D Breaux and Annie I Antón. Analyzing goal semantics for rights, permissions, and obligations. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 177–186, 2005. One citation in section 2.5.

[41] Travis D Breaux and Annie I Antón. Analyzing regulatory rules for privacy and security requirements. *Software Engineering, IEEE Transactions on*, 34(1):5–20, 2008. One citation in section 2.5.

[42] Travis D Breaux, Matthew W Vail, and Annie I Anton. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. pages 49–58, 2006. One citation in section 2.5.

[43] Travis D Breaux, Matthew W Vail, and Annie I Anton. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In *Requirements Engineering, 14th IEEE International Conference*, pages 49–58. IEEE, 2006. One citation in section 2.5.

[44] Christine Brentani. *Portfolio Management in Practice*. Elsevier, 2004. One citation in section 3.1.3.

[45] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004. One citation in section 2.2.1.

[46] John F P Bridges. Understanding the risks associated with resource allocation decisions in health: An illustration of the importance of portfolio theory. *Health, Risk & Society*, 6(3):257–275, 2004. One citation in section 3.1.2.

[47] S Brinkkemper. Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, 38(4):275–280, 1996. One citation in section 5.

[48] John Brondum and Liming Zhu. Visualising architectural dependencies. *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 7–14, 2012. One citation in section 4.3.

[49] Nanette Brown, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, Nico Zazworka, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, R Guo, R Ozkaya, and Kruchten. Managing technical debt in software-reliant systems. *Proceedings of the FSE/SDP workshop on Future of software engineering research, ser. FoSER '10*, page 47, 2010. 11 citations in sections 6, 1, 4.1.2, 1, 1, 2, 3, 4, 5, 9, and 4.3.

[50] Tyson R. Browning. On customer value and improvement in product development processes. *Systems Engineering*, 6(1):49–61, 2003. 2 citations in sections 2 and 3.

[51] Julien Brunel and Nora Cuppens-boulahia. Security Policy Compliance with Violation Management. pages 31–40, 2007. One citation in section 2.4.

[52] Brigitte Burgemeestre, Joris Hulstijn, YH H Tan, G Governatori, and G Sartor. Value-based argumentation for justifying compliance. *Deontic Logic in Computer Science*, pages 214–228, 2010. One citation in section 2.5.

[53] Frank Buschmann. To pay or not to pay technical debt. *IEEE Software*, 28(6):29–31, 2011. One citation in section 4.3.

[54] Antoine Cailliau and Axel Van Lamsweerde. A probabilistic framework for goal-oriented risk analysis. *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*, pages 201–210, sep 2012. One citation in section 2.2.1.

[55] David Callele. More than Requirements : Applying Requirements Engineering Techniques to the Challenge of Setting Corporate Intellectual Policy , An Experience Report 2011 Fourth International Workshop on. (Relaw):35–42, 2011. One citation in section 3.4.

[56] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt Och Dag. An industrial survey of requirements interdependencies in software product release plan-

ning. *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pages 84–92, 2001. One citation in section 2.3.1.

[57] Lawrence Chung and J Do Prado Leite. On Non-Functional Requirements in Software Engineering. *Conceptual modeling: Foundations and . . .*, pages 363–379, 2009. One citation in section 1.1.

[58] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012. One citation in section 1.1.

[59] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001. One citation in section 2.3.2.

[60] Zadia Codabux and Byron Williams. Managing technical debt: An industrial case study. *2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings*, pages 8–15, 2013. One citation in section 9.

[61] Robert G Cooper and Elko J. Edgett, Scott J & Kleinschmidt. Portfolio management for new product development : Results of an industry practices study portfolio management for new product development : results of an industry practices study. *R&D Management (Industrial Research Institute, Inc.)*, 13(4):1–39, 2001. One citation in section 3.1.2.

[62] Thierry Coq and Jean-Pierre Rosen. The SQALE quality and analysis models for assessing the quality of Ada source code. In *Reliable Software Technologies-Ada-Europe 2011*, pages 61–74. Springer, 2011. One citation in section 4.3.

[63] K A Crowe and W H Parker. Using portfolio theory to guide reforestation and restoration underclimate change scenarios. *Climatic Change*, 89(3-4):355–370, 2008. One citation in section 3.1.2.

[64] Ward Cunningham. The WyCash Portfolio Management System. *SIGPLAN OOPS Mess.*, 4(2):29–30, dec 1992. One citation in section 4.1.1.

[65] Bill Curtis, Jay Sappidi, and Alexandra Szynkarski. Estimating the principal of an application's technical debt. *IEEE Software*, 29:34–42, 2012. One citation in section 4.3.

[66] Bill Curtis, Jay Sappidi, and Alexandra Szynkarski. Estimating the size, cost, and types of technical debt. *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 49–53, 2012. One citation in section 4.3.

[67] Åsa G. Dahlstedt and Anne Persson. Requirements interdependencies: State of the art and future challenges. In *Engineering and Managing Software Requirements*, pages 95–116. Springer Berlin Heidelberg, 2005. One citation in section 5.4.1.

[68] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993. One citation in section 2.2.

[69] Jelle De Groot, Ariadi Nugroho, Thomas Bäck, and Joost Visser. What is the value of your software? In *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 37–44, 2012. 2 citations in sections 12 and 4.3.

[70] E Delarue, De Jonghe C, R Belmans, D W, Erik Delarue, Cedric De Jonghe, Ronnie Belmans, and D William. Applying Portfolio Theory To The Electricity Sector : Energy Versus Power Applying Portfolio Theory to the Electricity Sector : Energy versus Power. *Energy Economics*, (May), 2010. One citation in section 3.1.2.

[71] Leticia Duboc, Emmanuel Letier, and David S Rosenblum. Systematic Elaboration of Scalability Requirements through Goal-Obstacle Analysis. X:1–23, 2012. One citation in section 2.2.

[72] Edwin J Elton and Martin J Gruber. Modern portfolio theory, 1950 to date. *Journal of Banking & Finance*, 21(11-12):1743–1759, 1997. One citation in section 3.1.2.

[73] Neil A. Ernst. On the role of requirements in understanding and managing technical debt. *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 61–64, 2012. 2 citations in sections 5 and 4.3.

[74] Naeem Esfahani, Sam Malek, and Kaveh Razavi. GuideArch: Guiding the exploration of architectural solution space under uncertainty. In *Proceedings - International Conference on Software Engineering*, pages 43–52, 2013. 2 citations in sections 2.3.2 and 6.2.1.

[75] Frank J. Fabozzi, Harry M. Markowitz, Petter N. Kolm, and Francis Gupta. Mean-Variance Model for Portfolio Selection. In *Encyclopedia of Financial Models*. John Wiley & Sons, Inc., Hoboken, NJ, USA, dec 2012. One citation in section 3.1.3.

[76] Funmilade Faniyi, Rami Bahsoon, Andy Evans, and Rick Kazman. Evaluating security properties of architectures in unpredictable environments: A case for cloud. In *Proceedings - 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011*, pages 127–136, 2011. One citation in section 3.2.2.

[77] J Favaro. When the pursuit of quality destroys value [software development]. *IEEE Software*, 13:93–95, 1996. 2 citations in sections 2.3.3 and 2.3.3.

[78] John Favaro. Managing requirements for business value. *IEEE Software*, 19(2):15–17, 2002. 2 citations in sections 2.3.1 and 2.3.3.

[79] F Figge. Applying portfolio theory to biodiversity. *Biodiversity and Conservation*, 13(4):827–849, 2004. One citation in section 3.1.2.

[80] Francesca Arcelli Fontana, Vincenzo Ferme, and Stefano Spinelli. Investigating the impact of code smells debt on quality code evaluation. In *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 15–22, 2012. 2 citations in sections 2 and 4.3.

[81] Ralph Foorthuis and Rik Bos. A framework for organizational compliance management tactics. In *Lecture Notes in Business Information Processing*, volume 83 LNBIP, pages 259–268, 2011. One citation in section 2.4.

[82] Martin Fowler. Technical Debt Quadrant, 2009. 3 citations in sections (document), 4.1.3, and 4.1.

[83] Harris C Friedman. Real Estate Investment and Portfolio Theory. *Journal of Financial and Quantitative Analysis*, 6(2):861–874, 1971. One citation in section 3.1.2.

[84] Jesús García-Galán, Liliana Pasquale, George Grispos, and Bashar Nuseibeh. Towards adaptive compliance. In *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 108–114. ACM, 2016. One citation in section 2.5.

[85] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Security Requirements through Ownership, Permission and Delegation. *Proc. 13th IEEE Int'l Conf. Requirements Eng*, 2005. One citation in section 2.5.

[86] Martin Glinz. A risk-based, value-oriented approach to quality requirements. *IEEE Software*, 25(2):34–41, 2008. 2 citations in sections 2.3.1 and 2.3.3.

[87] Tony Gorschek and Magnus Wilson. Software Value Map. 2011. One citation in section 5.4.

[88] Daniel R. Greening. Release duration and Enterprise agility. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, pages 4835–4841, 2013. One citation in section 4.3.

[89] Paul Grünbacher, Alexander Egyed, and Nenad Medvidovic. Reconciling software requirements and architectures with intermediate models. *Software and Systems Modeling*, 3(3):235–253, 2003. One citation in section 2.4.5.

[90] Yuepu Guo, Carolyn Seaman, Rebeka Gomes, Antonio Cavalcanti, Graziela Tonin, Fabio Q B Da Silva, André L M Santos, and Clauirton Siebra. Tracking technical debt - An exploratory case study. *IEEE International Conference on Software Maintenance, ICSM*, pages 528–531, 2011. One citation in section 1.

[91] D Hartzell, John Hekman, and Mike Miles. Diversification categories in investment real estate. *Real Estate Economics*, (2):230–255, 1986. One citation in section 3.1.2.

[92] Sk Nahid Hasan, Mohammad Shabbir Hasan, Abdullah Al Mahmood, and Jahangir Alam. A Model for Value Based Requirement Engineering. *Journal of Computer Science*, 10(12):171–177, 2010. One citation in section 1.1.

[93] Wa Ël Hassan and Luigi Logrippo. Requirements and compliance in legal systems: a logic approach. *2008 Requirements Engineering and Law*, (d):40–44, sep 2008. One citation in section 2.5.

[94] Jeanette Heidenberg and Ivan Porres. Metrics functions for kanban guards. In *17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2010*, pages 306–310, 2010. 2 citations in sections 10 and 4.3.

[95] Andrea Herrmann and Maya Daneva. Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research. *2008 16th IEEE International Requirements Engineering Conference*, (iv):125–134, sep 2008. One citation in section 2.3.1.

[96] D J Hillier, Stephen A Ross, Randolph W Westerfield, Jeffrey Jaffe, and Bradford D Jordan. *Corporate finance.* Number 1st Eu. McGraw-Hill, 2010. One citation in section 5.

[97] Johannes Holvitie and Ville Leppanen. DebtFlag: Technical debt management with a development environment integrated tool. *2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings*, pages 20–27, 2013. One citation in section 2.

[98] Osterman Research Inc. Quantifying the Costs and Benefits of RSS in Perishables. *Agenda*, (August):297–306, 2001. 3 citations in sections 3, 2.3.2, and 5.4.

[99] Clemente Izurieta, Antonio Vetrò, Nico Zazworka, Yuanfang Cai, Carolyn Seaman, and Forrest Shull. Organizing the technical debt landscape. In *Proceedings of the Third International Workshop on Managing Technical Debt*, pages 23–26. IEEE Press, 2012. 2 citations in sections 5 and 4.3.

[100] Naseer Jan and Muhammad Ibrar. Systematic Mapping of Value-based Software Engineering-A Systematic Review of Value-based Requirements Engineering. *Engineering*, (December):1–232, 2010. One citation in section 2.3.1.

[101] Wayne Jansen and Timothy Grance. Guidelines on Security and Privacy in Public Cloud Computing. 2011. One citation in section 1.1.

[102] Wayne Jansen, Timothy Grance, et al. Guidelines on security and privacy in public cloud computing. *NIST special publication*, 800(144):10–11, 2011. One citation in section 1.

[103] J Karlsson, S Olsson, and K Ryan. Improved Practical Support for Large-scale Requirements Prioritising. *Requirements Engineering*, 1997. One citation in section 1.1.

[104] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, 1997. One citation in section 3.4.

[105] R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: a method for analyzing the properties of software architectures. *Proceedings of 16th International Conference on Software Engineering*, 1994. 2 citations in sections 2.3.2 and 2.4.5.

[106] Rick Kazman, Jai Asundi, and Mark H Klein. Design Decisions : An Economic Approach. *Software Engineering Institute Technical Report CMU/SEI-2002-TR-035*, (September):1–44, 2002. 2 citations in sections 2.3.2 and 2.4.5.

[107] Rick Kazman, Hoh Peter In, and Hong Mei Chen. From requirements negotiation to software architecture decisions. *Information and Software Technology*, 47(8):511–520, 2005. One citation in section 2.4.5.

[108] Rick Kazman, Mark Klein, and Paul Clements. ATAM : Method for Architecture Evaluation. *Cmusei*, 4(August):83, 2000. 2 citations in sections 2.3.2 and 2.4.5.

[109] Marwane El Kharbili, Qin Ma, Pierre Kelsen, Elke Pulvermueller, and Marwane El Kharbili. CoReL: Policy-Based and Model-Driven Regulatory Compliance Management. *2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, pages 247–256, aug 2011. One citation in section 2.4.1.

[110] Mahvish Khurum, Tony Gorschek, and Magnus Wilson. The software value map: an exhaustive collection of value aspects for the development of software intensive products. *Journal of Software: Evolution and Process*, 25(7):711–741, 2013. 5 citations in sections 5.4.1, 1, 2, 3, and 6.2.2.

[111] B. Kitchenham, S. Linkman, and D. Law. DESMET: a methodology for evaluating software engineering methods and tools. *Computing & Control Engineering Journal*, 8(3):120, 1997. 4 citations in sections 6.1, 6.1, 6.1, and 6.2.2.

[112] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, Stephen Linkman, and O Pearl Brereton. Systematic literature reviews in software engineering âÂĂÂŞ A systematic literature review. *Information and Software Technology*, 51(1):7–15, jan 2009. One citation in section 2.5.

[113] Barbara Kitchenham, Leslie Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *Software, IEEE*, 12(4):52–62, 1995. One citation in section 6.1.

[114] Tim Klinger, Peri Tarr, Patrick Wagstrom, and Clay Williams. An enterprise perspective on technical debt. *Proceeding of the 2nd working on Managing technical debt - MTD '11*, page 35, 2011. 2 citations in sections 2 and 8.

[115] Gerald Kotonya and Ian Sommerville. Requirements Engineering : Processes and Techniques (Worldwide Series in Computer Science). *Star*, page 294, 1998. One citation in section 1.1.

[116] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice, nov 2012. 3 citations in sections 1, 10, and 1.

[117] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. *Proceedings of 5th IEEE International Symposium on Requirements Engineering*, (August), 2001. 11 citations in sections 1.1, 3, 4, 2.2, 2, 3, 4, 5, 6, 7, and 5.3.1.

[118] Axel Van Lamsweerde. Risk-Driven Engineering of Requirements for Dependable Systems. *Engineering Dependable Software Systems*, pages 1–36, 2013. One citation in section 2.2.1.

[119] Alexei Lapouchnian. Goal-Oriented Requirements Engineering : An Overview of the Current Research. *Requirements Engineering*, 8(3):32, 2005. One citation in section 2.2.

[120] Emmanuel Letier, Axel Lamsweerde, a. van Lamsweerde, and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000. 8 citations in sections 5, 1, 2, 3, 2.2, 2.2.1, 5.4.3, and 5.4.3.

[121] Emmanuel Letier, David Stefan, and Earl T. Barr. Uncertainty, risk, and information value in software requirements and architecture. *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, pages 883–894, 2014. 3 citations in sections 1.1, 2.2.1, and 2.3.2.

[122] Emmanuel Letier and Axel Van Lamsweerde. Deriving operational software specifications from system goals. *ACM SIGSOFT Software Engineering Notes*, 27(6):119, 2002. 3 citations in sections 2, 3, and 5.

[123] Jean Louis Letouzey. The SQALE method for evaluating technical debt. In *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 31–36, 2012. One citation in section 6.

[124] Jean Louis Letouzey and Michel Ilkiewicz. Managing technical debt with the SQALE method. *IEEE Software*, 29:44–51, 2012. 3 citations in sections 13, 6, and 4.3.

[125] Z Li, N Guelfi, P Liang, P Avgeriou, and a Ampatzoglou. An empirical investigation of modularity metrics for indicating architectural technical debt. *QoSA 2014 - Proceedings of the 10th International ACM SIGSOFT Conference on Quality of Software Architectures (Part of CompArch 2014)*, pages 119–128, 2014. One citation in section 11.

[126] Zengyang Li, Paris Avgeriou, and Peng Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220, 2015. 4 citations in sections 7, 8, 4.1.5, and 4.2.1.

[127] H Markowitz. Portfolio Selection: Efficient Diversification of Investments. 1957. 5 citations in sections 3.1.1, 3.1.2, 3.2.3, 3.3, and 3.3.4.

[128] Marriam-Webster Dictionary. *No Title*. One citation in section 2.3.

[129] Aaron Keith Massey. *Legal Requirements Metrics for Compliance Analysis*. PhD thesis, 2012. One citation in section 2.5.

[130] AK K Massey, PN N Otto, A I AntÃÂşn, and AI Antón. Prioritizing legal requirements. *Requirements Engineering and . . .*, 1936(2009), 2009. One citation in section 2.5.

[131] Jeremy C. Maxwell and Annie I. Anton. Developing Production Rule Models to Aid in Acquiring Requirements from Legal Texts. *2009 17th IEEE International Requirements Engineering Conference*, pages 101–110, aug 2009. One citation in section 2.5.

[132] Jeremy C Maxwell and Annie I Anton. Checking Existing Requirements for Compliance with Law Using a Production Rule Model. pages 0–5, 2010. One citation in section 2.5.

[133] M May, C Gunter, and I Lee. Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies. *19th IEEE Computer Security Foundations Workshop*, 2006. 2 citations in sections 1.1 and 2.5.

[134] Alois Mayr, Reinhold Plosch, and Christian Korner. A benchmarking-based model for technical debt calculation. *Proceedings - International Conference on Quality Software*, pages 305–314, 2014. One citation in section 4.2.1.

[135] Steve McConnell. Managing technical debt (talk), 2013. 2 citations in sections 4.1.3 and 4.2.1.

[136] J. Yates Monteith and John D. McGregor. Exploring software supply chains from a technical debt perspective. In *2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings*, pages 32–38. IEEE Computer Society, 2013. One citation in section 4.3.

[137] M. Moore, R. Kaman, M. Klein, and J. Asundi. Quantifying the value of architecture design decisions: lessons from the field. *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 557–562, 2003. One citation in section 5.4.1.

[138] J. David Morgenthaler, Misha Gridnev, Raluca Sauciuc, and Sanjay Bhansali. Searching for build debt: Experiences managing technical debt at Google. In *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 1–6, 2012. One citation in section 2.

[139] Md Rashed I. Nekvi and Nazim H. Madhavji. Impediments to regulatory compliance of requirements in contractual systems engineering projects: A case study. *ACM Trans. Manage. Inf. Syst.*, 5(3):15:1–15:35, December 2014. One citation in section 2.5.

[140] Rashed Nekvi, Remo Ferrari, Brian Berenbach, and Nazim H. Madhavji. Towards a compliance meta-model for system requirements in contractual projects. *2011 Fourth International Workshop on Requirements Engineering and Law*, (Relaw):74–77, aug 2011. One citation in section 2.5.

[141] Robert L. Nord, Mario R. Barbacci, Paul Clements, Rick Kazman, Mark Klein, Liam O'Brien, and James E. Tomayko. Integrating the Architecture Tradeoff Analysis Method (ATAM) with the cost benefit analysis method (CBAM). *Sei - Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Institute*, (December):1–35, 2003. One citation in section 5.4.

[142] Robert L. Nord, Ipek Ozkaya, Philippe Kruchten, and Marco Gonzalez-Rojas. In search of a metric for managing architectural technical debt. *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012*, pages 91–100, 2012. 2 citations in sections 6 and 4.3.

[143] Ariadi Nugroho, Joost Visser, and Tobias Kuipers. An empirical model of technical debt and interest. *MTD '11: Proceedings of the 2nd Workshop on Managing Technical Debt*, pages 1–8, 2011. 2 citations in sections 2 and 4.3.

[144] Emil Numminen. Why Software Platforms Make Sense in Risk Reduction in Software Development–A Portfolio Theory Approach. In *Proceedings of the 4th European Conference on Information Management and Evaluation*, 2010. 2 citations in sections 3.1.2 and 3.4.

[145] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. *ICSE-2000 The Future of Software Engineering*, 2000. One citation in section 5.3.1.

[146] Paul N. Otto and Annie I. Antoon. Addressing legal requirements in requirements engineering. *Proceedings - 15th IEEE International Requirements Engineering Conference, RE 2007*, pages 5–14, 2007. One citation in section 1.1.

[147] Ipek Ozkaya, Philippe Kruchten, Robert L. Nord, and Nanette Brown. Managing technical debt in software development. *ACM SIGSOFT Software Engineering Notes*, 36(5):33, sep 2011. One citation in section 4.1.1.

[148] Michael P. Papazoglou. Making business processes compliant to standards & regulations. In *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC*, pages 3–13, 2011. One citation in section 2.4.2.

[149] Birgit Penzenstadler. Towards a Definition of Sustainability in and for Software Engineering. In *28th Annual ACM Symposium on Applied Computing*, pages 1183–1185, 2013. One citation in section 6.2.1.

[150] Pamela Peterson Drake and Frank J. Fabozzi. *The basics of finance an introduction to financial markets, business finance, and portfolio management.* Wiley, 2010. 5 citations in sections 3.1.1, 2, 3, 4, and 5.

[151] B. J. Oates P. Brereton M. Azuma R. Dawson, P. Bones, B. J. Oates P. Brereton M. Azuma M. L. JacksonR. Dawson, P. Bones, and M. L. Jackson. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131 – 164, 2008. One citation in section 6.1.

[152] Narayan Ramasubbu and Chris F. Kemerer. Towards a model for optimizing technical debt in software products. *2013 4th International Workshop on Managing Technical Debt (MTD)*, pages 51–54, 2013. One citation in section 4.1.2.

[153] L Saaty. The Analytical Hierarchy Process. 1980. One citation in section 3.4.

[154] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer.* John Wiley & Sons, 2008. One citation in section 6.3.2.

[155] Jay Sappidi, Bill Curtis, and Alexandra Szynkarski. The CRASH Report - 2011/12 Summary of Key Findings. Technical report, 2011. One citation in section 6.

[156] P Sawyer, B Paech, P Heymans, Aybüke Aurum, and Claes Wohlin. A Value-Based Approach in Requirements Engineering: Explaining Some of the Fundamental Concepts. In *Proceedings 13th International Working Conference on Requirements Engineering: Foundation for Software Quality*, number June, pages 109–115, 2007. 5 citations in sections 2.3, 1, 2, 3, and 2.3.1.

[157] Klaus Schmid. A formal approach to technical debt decision making. *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures - QoSA '13*, page 153, 2013. One citation in section 1.

[158] Klaus Schmid. Technical debt — from metaphor to engineering guidance: A novel approach based on cost estimation. Informatikberichte 1/2013, SSE 1/13/E, Institute of Computer Science, University of Hildesheim, 2013. One citation in section 4.3.

[159] Carolyn Seaman and Yuepu Guo. A portfolio approach to technical debt management. *Proceedings - International Conference on Software Engineering*, 82:31–34, 2011. 2 citations in sections 1 and 4.3.

[160] Carolyn Seaman and Yuepu Guo. *Measuring and Monitoring Technical Debt*, volume 82. 2011. 3 citations in sections 4, 4, and 4.3.

[161] Carolyn Seaman, Yuepu Guo, Clemente Izurieta, Nico Zazworka, Forrest Shull, Antonio Vetrò, and A Vetro. Using technical debt data in decision making: Potential decision approaches. *Third International Workshop on Managing Technical Debt (MTD)*, pages 45–48, 2012. One citation in section 1.

[162] Zahra Hossein Abad Shakeri and Guenther Ruhe. Using Real Options to Manage Technical Debt in Requirements Engineering. *IEEE International Requirements Engineering Conference*, 23:230–235, 2015. One citation in section 10.

[163] William F Sharpe. The sharpe ratio. *The journal of portfolio management*, 21(1):49–58, 1994. One citation in section 4.2.2.

[164] Alberto Siena, F B K Irst, Anna Perini, John Mylopoulos, and Angelo Susi. Towards a framework for law-compliant software requirements. *31st Int. Conf. on Software Engineering (ICSE'09).*, pages 251–254, 2009. 5 citations in sections 2.4, 1, 3, 2.5, and 6.

[165] Alberto Siena, John Mylopoulos, Anna Perini, and Angelo Susi. Designing law-compliant software requirements. In *Lecture Notes in Computer Science (including*

subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 5829 LNCS, pages 472–486, 2009. 6 citations in sections 2.4, 7, 1, 2, 6, and 7.

[166] S Sivzattian and B Nuseibeh. Linking the Selection of Requirements to Market Value: A Portfolio-Based Approach. 2 citations in sections 3.1.2 and 3.4.

[167] Will Snipes, Brian Robinson, Yuepu Guo, and Carolyn Seaman. Defining the decision factors for managing defects: A technical debt perspective. In *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, pages 54–60, 2012. 4 citations in sections 9, 4, 3, and 4.3.

[168] Ian Sommerville. Software Engineering. *Empirical Software Engineering*, 10(May 2000):251–253, 2005. One citation in section 1.1.

[169] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. 1997. One citation in section 1.1.

[170] David Stefan and Emmanuel Letier. Supporting Sustainability Decisions in Large Organisations. *Proceedings of the 2014 conference ICT for Sustainability*, (Ict4s):333–341, 2014. One citation in section 5.1.

[171] Kevin J Sullivan, Prasad Chalasani, Somesh Jha, and Vibha Sazawal. Software design as an investment activity: a real options perspective. *Real options and business strategy: Applications to decision making*, pages 215–262, 1999. One citation in section 7.3.1.

[172] Kevin J Sullivan, John Socha, and Mark Marchukov. Using formal methods to reason about architectural standards. In *Proceedings - International Conference on Software Engineering*, pages 503–513, 1997. One citation in section 4.

[173] Marcin Szlenk, Andrzej Zalewski, and Szymon Kijas. Modelling Architectural Decisions Under Changing Requirements. In *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, WICSA-ECSA '12, pages 211–214, Washington, DC, USA, 2012. IEEE Computer Society. One citation in section 1.1.

[174] Marcin Szlenk, Andrzej Zalewski, and Szymon Kijas. Modelling architectural decisions under changing requirements. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 211–214. IEEE, 2012. One citation in section 2.

[175] Edith Tom, Aybüke Aurum, and Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516, 2013. 9 citations in sections 1, 3, 5, 6, 8, 13, 6, 9, and 4.2.1.

[176] Richard Torkar. Adopting Free/Libre/Open Source Software Practices, Techniques and Methods for Industrial Use. *Journal of the Association for Information Systems*, 12(1):88–122. 35p. 5 Diagrams, 2011. One citation in section 7.

[177] N Ur Rehman, S Bibi, S Asghar, and S Fong. Comparative study of Goal-Oriented Requirements Engineering. *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*, pages 248–253, 2010. One citation in section 2.2.

[178] a. Van Lamsweerde and E. Letier. Integrating obstacles in goal-driven requirements engineering. *Proceedings of the 20th International Conference on Software Engineering*, pages 53–62, 1998. One citation in section 5.4.3.

[179] Axel Van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998. 2 citations in sections 4 and 2.2.

[180] J Vanwelkenhuysen. Quality requirements analysis in customer-centered software development. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on*, pages 117–124, 1996. One citation in section 1.1.

[181] Michael R Walls. Combining decision analysis and portfolio management to improve project selection in the exploration and production firm BT - Risk Analysis Applied to Petroleum Exploration and Production. *Journal of Petroleum Science and Engineering*, 44(1-2):55–65, 2004. One citation in section 3.1.2.

[182] Jun Wang and Jianhan Zhu. Portfolio theory of information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '09*, pages 115–122. ACM Press, 2009. One citation in section 3.1.2.

[183] Kristian Wiklund, Sigrid Eldh, Daniel Sundmark, and Kristina Lundqvist. Technical Debt in Test Automation. *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 887–892, 2012. One citation in section 12.

[184] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*, volume 15. 2000. One citation in section 6.2.4.

[185] World Commission on Environment and Development. Report of the World Commission on Environment and Development: Our Common Future (The Brundtland Report). Technical Report 1, UN, 1987. One citation in section 6.2.1.

[186] Ernst & Young. 10th Annual Global Information Security Survey. Technical report, 2005. One citation in section 1.1.

[187] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997. One citation in section 2.2.

[188] Nico Zazworka, Carolyn Seaman, and Forrest Shull. Prioritizing design debt investment opportunities. *Proceeding of the 2nd working on Managing technical debt - MTD '11*, page 39, 2011. One citation in section 4.3.

[189] Nico Zazworka, Rodrigo O. Spínola, Antonio Vetro', Forrest Shull, and Carolyn Seaman. A case study on effectively identifying technical debt. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering - EASE '13*, page 42. ACM Press, 2013. 2 citations in sections 3 and 5.

[190] Nico Zazworka, Antonio Vetro', Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Seaman, and Forrest Shull. Comparing four approaches for technical debt identification. *Software Quality Journal*, pages 1–24, 2013. 2 citations in sections 5 and 4.3.