Quantitative Assessment of Nonfunctional Requirements in Product Families

QUANTITATIVE ASSESSMENT OF NONFUNCTIONAL REQUIREMENTS IN PRODUCT FAMILIES

BY

REHAM FADUL, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2014) McMaster University

(Software Engineering) Hamilton, Ontario, Canada

TITLE:	Quantitative Assessment of Nonfunctional Requirements in Product Families
AUTHOR:	Reham Fadul B.Eng., (Computer Science) Umm Al-Qura University, Makkah, Saudi Arabia
SUPERVISOR:	Ridha Khedri and Tom Maibaum
NUMBER OF PAGES:	x, 96

To the incredible woman who brought me to this life, to my source of happiness, and the love of my life, my mother **Amnah Hemaidah**. *To my father* **Elias Fadul**, *whom without his prayers I would not be the person I am today. To whom I share with my childhood memories and grown-up dreams, my siblings,* **Alaa**, **Walaa**, **Rahaf**, **Ahmad**, *and* **Aziz**. *To my little angels who spread joy and fun all over our lives, my nephews* **Battal** *and* **Moayad**, *and my niece* **Aleene**. *Last but not least, to my best friend, my sister from another mister, and my companion throughout this journey,* **Nuha Zamzami**. *To you and to all the people who love me, I say that you are amazing and I just could not imagine my life without you. I would have never done this without your love and support. God Bless you all. I am so proud to have you in my life and I hope that I have made you as proud.*

I dedicate this to all of you ...

Love,

Reham

Abstract

Modelling nonfunctional requirements, along with many other NFR-related concerns, have not been addressed properly in the literature. Although nonfunctional requirements (or quality attributes) are important, they are the most expensive and difficult to deal with since they are (mostly) specified qualitatively not quantitatively, and also due to the fact that nonfunctional requirements may have interdependencies among each other leading to inconsistency in requirements specification. Moreover, the adoption of the concept of product families into the software industry led to the ability today to build families that share features. This indicates the significance of software families. Accordingly, in this research, not only do we study nonfunctional requirements in a systematic way, we also attempt to examine them from the perspective of software families. We highlight the need for developing a better quantitative assessment technique for nonfunctional requirements. Then, we propose a formal approach to the assessment of nonfunctional requirements in software product families.

Acknowledgements

I would like to thank my supervisors, Prof. Ridha Khedri and Prof. Tom Maibaum, for the patient guidance, encouragement and advice they have provided throughout my time as their student. I am as well grateful to Jason Jaskolka who helped me during my supervisor's absence.

Finally, I would like to thank the Ministry of Higher Education in my home country, The Kingdom of Saudi Arabia, not only for providing the funding which allowed me to undertake this research, but also for giving me the opportunity to attend conferences and meet so many interesting knowledgable people.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1 General Context

Requirement's Engineering (RE) plays a crucial fundamental rule in Software Engineering (SE). It is concerned with the gathering, documentation, and evaluation of the requirements of the system to be. Zave [Zav97] writes:

> Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families.

We have two types of software system requirements to deal with: Functional Requirements (FRs) and Nonfunctional Requirements (NFRs). A functional requirement articulates the behaviour of the system to be. This includes calculations, data manipulation, processing, and any other activities related to what the system must accomplish.

As defined by IEEE [SCCotICS90], a functional requirement is “a requirement that specifies a function that a system or system component must be able to perform“. For instance, a stakeholder’s functional requirement for a library system may include the specification of a checkout function for borrowing books after checking the maximum number of books allowed to be borrowed by a user. Therefore, checking their satisfaction is straightforward, either the function is performed by the system or not.

When implementing functional requirements, nonfunctional requirements are needed to impose constraints on the implementation. Thus, functional requirements alone are not enough to produce a reliable independent system, which means that functional and nonfunctional requirements are of complementary nature.

A wide range of concerns falls under the name of nonfunctional requirements such as reliability, security, accuracy, safety, and performance. They express constraints that need to be satisfied by functional requirements. These concerns form as well the overall quality of the system to be; therefore they are called quality requirements, quality attributes, and quality characteristics of the system. As illustrated in [Tha02], a nonfunctional requirement is “a software requirement that describes not what the software will do, but how the software will do it“.

Nonfunctional requirements are also vague and often subjective in nature. By definition, they are not related to any particular function and often specified in an imprecise way, leading different stakeholders to interpret them differently [GW01]. Hence, they could have different meanings to different people in different environments. For example, what some see as a high performance system, others may see it as a low performance system. This is due to the absence of an exact measure for what high performance is and it is usually based on the former experience of the user. System developers may consider a complex information system with a response time of 1-2 seconds as efficient. However, users who lack the technical backgrounds may not agree [BKJ09].

For the purpose of differentiating, a functional requirement is a certain task, service, or process that must be performed by the system, while the nonfunctional requirement is the quality of service standard that a system must cope with in order to build that standard into the system.

2 Analysis of Nonfunctional Requirements

Nonfunctional requirements present interdependencies among each other. These interdependencies can be cooperative, conflicting, irrelevant, or counterbalance [LK98]. We find in [BKJ09] a list of different interdependency relationships: EQUAL, MAKES, SOME+, HELPS, UNKNOWN, HURTS, SOME-, or BREAKS.

A nonfunctional requirement R_1 MAKES for R_2 when an increase in R_1's satisfaction is accompanied by an increase in the satisfaction of R_2. An example is the direct relationship between reliability and security. Hence, we can say that security MAKES for reliability.

Vice versa, when an increase of R_1's satisfaction leads to a decrease in R_2's satisfaction, it is said that R_1 and R_2 have a negative dependency: HURTS. Then, the two requirements are said to be conflicting. For instance, the flexibility and efficiency of the C-related programming languages in providing low-level access to the hardware machine is accompanied by the lack of safety measures. Such conflicts usually leads the functionalities that help satisfy one nonfunctional requirement to hinder another requirement. The fact that nonfunctional requirements are often conflicting is one of the reasons why nonfunctional requirements are difficult to assess and to manage.

Since nonfunctional requirements have both positive and negative impacts among each other, then they need to be dealt with separately [CdPL04]. This separation of concerns is done through a conflict-resolution technique called the trade-off analysis. The analysis of nonfunctional requirements is one of the most important activities in requirement engineering. In many cases, analysis methodologies are based on a translation of the software model into a different notation to be analyzed [ES08]. For example, in scenario-based analysis, a requirement must be transformed into a scenario in order to be analyzed.

In requirement engineering analysis, functional requirements are analyzed through decomposition into further specifications, whereas analysis of nonfunctional requirements can be carried out by their traceability to a design or by their assessment with an implemented system [SM98]. On one hand, if we have a design and we want to ensure the satisfiability of that design to its nonfunctional requirements, we use techniques to justify design decisions on the inclusion or exclusion of nonfunctional requirements that impact on that software design [RG11]. On the other hand, assessing nonfunctional requirements against an implemented system measures how much a software system is in accordance with the set of nonfunctional requirements that it should satisfy [RG11]. The latter set of approaches is concerned with assessing nonfunctional requirements after the development process while the earlier set is concerned with satisficing nonfunctional requirements while the making of design decisions.

Modelling approaches of nonfunctional requirements have two general classifications, process-oriented versus product-oriented, and qualitative versus quantitative. Product-oriented approaches measure the software quality criteria once a product is built. Process-oriented approaches focus on nonfunctional requirements and their interactions to make design decisions. Qualitative approaches study the relationships between nonfunctional requirements and reason about their trade-offs qualitatively. Quantitative approaches find measures to calculate the satisfaction of quality attributes.

The literature is enriched with many modelling techniques for nonfunctional requirements. Some of these works are goal oriented, which addresses nonfunctional requirements as a first modelling concept. They represent them as vague not sharply defined soft-goals and then decompose them into more well-defined goals. This kind of models represent relationships between these soft-goals as well to ease detecting conflicts or cooperations between goals and their refinements. The satisfaction of these models are qualitative and assessed using some labelling procedure. Examples of this kind of models are the *Chung NFR-Framework* and *Soft-goal Interdependency Graphs (SIGs).* The NFR-Framework minds the gap between nonfunctional requirements and design. However, it lacks any quantitative support towards analyzing and assessing them.

Another kind of models is the strategic dependency models such as the $i*$ *family.* In the $i*$ structure, actors are the main modelling concept. Relationships between these actors are captured in an organizational setting, and then the system's goals are related to this organizational setting.

Moreover, we have the *scenario-based technique.* It basically identifies nonfunctional requirements and creates a scenario for each one of them. These scenarios will later on be transformed into an executable format to be executed by a prototype or a model of the system. This technique is actually a validation and testing technique rather than a modelling one for nonfunctional requirements.

A major drawback of these techniques is that they model nonfunctional requirements on the basis of the overall former experience of the software architect. Not to mention that the assessment is mostly qualitative. Another drawback is that they are mostly graphical representations that are very limited in use when expanding the system model. Also, they can only model nonfunctional requirements in single products and cannot be used in product families. This contradicts with the interests of many organizations that adopt the concept of software product families.

Moreover, the literature contains some quantitative modelling techniques that address the issue of quantifying nonfunctional requirements. For instance, the Requirement Hierarchy Approach (RHA) [Rya00] quantifies quality requirements by decomposing them into quantifiable functional requirements. This approach completely ignores the fact that nonfunctional requirements are not always decomposed into functional requirements. Another example is a model proposed in [GS13] to evaluate nonfunctional requirements quantitatively in Software Product Lines (SPL). However, this model actually evaluates the processes carried by behavioural models (variable sequence diagrams) and not the feature models of the system.

Given all these quantitative models, the problem is still unsolved. The question here is why not to have a quantitative evaluation for the satisfaction of nonfunctional requirements in a certain system? This quantitative evaluation can be founded on the partial quantitative evaluation of each and every system component to these nonfunctional requirements. This research proposes such a model.

3 Problem Statement

Modelling nonfunctional requirements, along with many other NFR-related concerns, have not been addressed properly in the literature. One of the reasons is that although nonfunctional requirements are important, they are the most expensive and difficult to deal with for a number of reasons. Firstly, they are specified qualitatively not quantitatively. Secondly, a nonfunctional requirement (or a quality attribute) may have a negative interdependency with another quality attribute leading to inconsistency in requirement specifications. The conflict resolution is qualitative as well. Also, software quality attributes have a tendency of being imprecise and subjective, therefore precise methods and tools are needed to create software products and services [JFS06].

The concept of commonality and variability in product families adds an additional dimension of complexity. It is unknown how this concept affects the modelling of nonfunctional requirements, and it ought to be investigated further. An example of carrying the variability notion of product families into nonfunctional requirements is the following. On one hand, a nonfunctional requirement may be required by all products within a software family such as security. However, a requirement can also be demanded in varying levels among families such as intense security or moderate level of security. On the other hand, some families may need a certain nonfunctional requirement, while others do not.

In this research, issues and challenges regarding modelling nonfunctional requirements will be discussed, and then extended to software product families. However, the main objective will be conducting a model for quantitative assessment of nonfunctional requirements. We will also discuss the ability to articulate this model to capture quantitative assessment of nonfunctional requirements within a software product family.

4 Motivation

According to Zave's statement in page 1, it is an essential part of requirement engineering to study its evolvement over software families. Thus, not only do we have to study nonfunctional requirements in a systematic way, but also we have to examine it from the perspective of software families. Due to the adoption of the concept of product families into the software development, today we build families that share features [HKM11]. This way has become very common in the software industry world, which also indicates the significance of software families in our study. Particularly, we need to seek documenting, analyzing, and modelling of nonfunctional requirements in product families.

Zave's work [Zav97] is quite old, but the situation did not really change. According to Nguyen [Ngu09], the literature still does not provide a systematic way to analyze and design nonfunctional requirements from the perspective of the concept of commonality and variability of product families.

Moreover, the quality of product families is determined by the satisfaction of their nonfunctional requirements (quality attributes). However, the evaluation of project quality attributes is one particular difficult problem. It highly depends on the experiences of software architects. Quantifying quality attributes is feasible in many cases. An example is when defining the performance attribute of an online media broadcast server [TFQ06] where many factors can be defined and evaluated quantitatively (i.e., bandwidth and throughput). Even though we almost have some sort of quantitative frameworks, we are still far away from being fully capable of assessing architectural designs quantitatively. In order to solve the problem, a software architect must answer the following two questions [TFQ06]: what is the possibility of satisfying the quality attributes of the proposed architecture design? And how to establish the NFR-dependencies among system's components?

Quantitative approaches either formally estimate the probability of failure in meeting and satisfying the quality requirements, or informally yet rigorously specify the degree to which software properties are contributing towards the satisfaction of same qualities [JFS06]. A formal method is a technique based on mathematics to model complex systems as mathematical entities. We intend to ground the model by formal methods in order to be able to reason about the satisfaction of nonfunctional requirements quantitatively and have values for the quality of products.

5 Thesis Contribution

This thesis presents a mathematical model for a quantitative assessment of nonfunctional requirements in product families. We aim through the construction of this model to analyze nonfunctional requirements in an automated fashion, to measure the impact caused by nonfunctional requirements over software product families, and to have a well-established way to answer numerous questions in regards of the numerical analysis and modelling of nonfunctional requirements. For example, such questions may include the overall quantitative assessment of the impact caused by two nonfunctional requirements (i.e., performance and security) simultaneously over a product family.

We set out to develop this model using some mathematical structures including the algebraic mathematical structure of Γ-*semirings* and *intervals*. The choice of these mathematical structures is suitable towards modelling nonfunctional requirements for reasons that will be illustrated later on in Chapter 4. Our model is specifically tackling nonfunctional requirements in product families.

6 Thesis Organization

In Chapter 2, we review closely some of the NFR-modelling techniques and frameworks, and then we introduce product family algebra and its basics. In Chapter 3, we take a closer look at some mathematical backgrounds that are necessary for the proposed approach. In Chapter 4, we present in details the mathematical model for the quantitative assessment of nonfunctional requirements in product families. In Chapter 5, we conclude and lay some potential future work.

Chapter 2

NFRs Modelling Techniques

The acceptance of a software product by a customer highly depends on its overall quality [RG11]. These overall quality attributes are commonly referred to as nonfunctional requirements. The incorporation of a quality in a product can be done systematically upon the consideration of the dependencies among NFR goals and functional features [PLZ09]. Therefore, identifying nonfunctional requirements is vital. A way to achieve that is through requirement analysis.

Several analysis models for requirements are presented in the literature [UK11]. Some early works [AK12] [MCN92] on nonfunctional requirements are *product-oriented* as they measure how much a software system is in accordance with a set of nonfunctional requirements that it should satisfy. Therefore, they are more like evaluation tools that take place at the end of the software development life cycle to assure the satisfaction of quality requirements. In contrast, newer works are *process-oriented* that take place at each stage of the development cycle to aid the decision-making process of the inclusion of each nonfunctional requirement. The latter kind of works explicitly deal with nonfunctional requirements [RG11] [BKJ09].

Nonfunctional requirements are always expressed in a general abstract manner that is later on refined into more details [CdPL09], which makes goal-oriented models suitable for modelling these requirements. Goal-oriented (or goal-centric) models emphasize relationships between nonfunctional requirements and treat them in depth. Some goal-oriented models, such as NFR-framework, KAOS [DvLF93], and i^* family [Yu97] deal with nonfunctional requirements as a first modelling concept merged together with functionality [CdPL09].

For the sake of a better treatment of nonfunctional requirements, a classification of process-oriented approaches was proposed in [JFS06]. These approaches help reasoning about quality requirements during the early stages of the development process. The approaches are classified as follow: formal, informal, qualitative, and quantitative. Formal approaches are mathematical approaches that use formal notations such as temporal logic or fuzzy logic to specify nonfunctional requirements in a precise accurate way [JFS06]. Semi-formal or informal approaches use structured non-mathematical notations to organize information about nonfunctional requirements [JFS06]. Qualitative approaches use qualitative measures and traditional labels to specify the degree of satisfaction of quality requirements [JFS06]. Finally, quantitative approaches either formally estimate the probability of failure in meeting and satisfying the quality requirements, or informally specify the degree to which software properties are contributing towards the satisfaction of same qualities [JFS06].

In this chapter, we survey some of the process-oriented and goal-oriented models for assessing nonfunctional requirements within a product or a family of products.

1 Informal Qualitative Modelling Techniques

1.1 Chung's NFR-Framework and Soft-Goal Interdependency Graphs SIGs

Chung et al. [CdPL09] proposed a qualitative approach for dealing with nonfunctional requirements. It is called the *NFR-framework* and it is process-oriented in the sense that it takes place at each stage of the development process. The NFR-framework promotes goal orientation, but with the main emphasis on nonfunctional requirements. As stated by Chung [CdPL09], the idea behind the NFR-framework is the *Soft-goal Interdependency Graphs (SIGs)*, which makes requirement analysis one of many areas in software engineering that have dealt with some aspects of SIGs. Basically, the NFR-framework is the proposed model whereas SIG is a visual graphical representation of the model that maintains soft-goals and their interdependencies.

SIGs [BKJ09][CBLA08] describe both nonfunctional requirements and their alternative solutions. When defining a nonfunctional requirement, it is inevitable to use the definition of another nonfunctional requirement, which leads to an increased complexity in the original nonfunctional requirement's satisfaction. Therefore, nonfunctional requirements should not be addressed absolutely, and here emerges the need to use the concept of soft-goals. *Soft-goals* are goals with success criteria not sharply defined [CBLA08]. If the criteria are met to an acceptable level, then the soft-goal is considered satisficed rather than satisfied. There are mainly three types of soft-goals in SIGs: *NFR soft-goals*, *operationalizing soft-goals*, and *claim soft-goals*. The NFR soft-goals are the nonfunctional requirements to be satisficed. The operationalizing soft-goals or hard-goals are the functional requirements that help satisficing the NFR soft-goals. Finally, the claim soft-goals describe the rationale of soft-goals or a refinement between them.

Interdependencies between nonfunctional requirements and their refinements are also shown in SIGs. There are multiple kinds of correlations (or, interdependencies) that indicate the rationale of all different relationships between goals. These correlations are: *Break* '$-$' and *Hurt* '$--$' for negative relationships, *Unknown* '?' for undefined relationships, *Help* '+' and *Make* '++' for positive relationships. The hard-goals, soft-goals, and correlations all together conclude the major elements of SIGs.

The analysis process goes as follows. We start with a high-level set of NFR soft-goals that are decomposed into operationalizing soft-goals. The vagueness and level of ambiguity of the NFR soft-goals decreases gradually as the decomposition process continues until we reach clearly defined operationalizing soft-goals.

SIGs can be visualized graphically as follows. NFR soft-goals are used to represent the stakeholder's objectives and therefore they correspond graphically to the roots of the graph. They need to be further analyzed and refined throughout AND/OR decomposition trees into operationalization sub-goals that correspond graphically to the leaf nodes of the graph. These operationalizations can be either *mandatory* or *optional*. Mandatory operationalizations are must to include, whereas optional ones are included up to the choice of the software architect. OR-decomposition between NFR-soft-goals and its operationalizations represent alternative and optional operationalizations, while AND-decomposition represents all included mandatory operationalizations. The degree to which each operationalization is willing to contribute towards the satisfaction of their parents is qualitatively defined and assigned to the operationalizations using some labelling system. Thus, edges are labeled by the characterizations of previously mentioned correlations (i.e., $--, -, ?, +, ++$). This is made to clarify both positive and negative interdependencies between hard-goals and soft-goals. Thus, conflicts can be identified between operationalizations.

The refined operationalizations are the system's potential functionalities and they provide different alternative solutions for satisficing the parent NFR soft-goals. Some of the alternative solutions are of complementary nature and others are of conflicting nature. In case of compatibility, more than one solution can be chosen to satisfice the parent NFR soft-goal [CBLA08]. Otherwise, the software architect has to choose the most suitable one. The process of choosing the most suitable alternative is called the *trade-off analysis*, and it is directed and run by the software architect. SIGs enable software architects to clearly describe all options and their relationships.

1.2 A Process-Oriented Approach for Representing Nonfunctional Requirements

A comprehensive framework was proposed in [MCN92] for representing NFRs. The framework is process-oriented in terms that it takes place during the development process to assess the design-decision making at each stage of the process. The approach is mainly composed of the next four components: *goals*, *link types*, *methods*, and *labelling procedure*.

Goals: Since the goals representing nonfunctional requirements cannot be satisfied absolutely, it is suggested to refer to their satisfaction status as satisficed when a goal is satisfied and met within an acceptable limit. There are three goal classes. Goals representing nonfunctional requirements called *NFR-goals*, goals representing design decisions called *satisficing goals*, and goals representing arguments against or in support of other goals called *argumentation goals*. All these goals are organized in a specialization/generalization manner using a traditional AND/OR tree. Each goal has a *sort* associated to it. The sorts of NFR-goals vary among the different categories of nonfunctional requirements such as performance and security. Whereas the sorts of satisficing goals range over the different categories of design decisions. Some sorts can be further sub classified into sub-sorts. For instance, the performance sort can be subdivided into time performance and space performance sub-sorts. Nevertheless, the argumentation goals have fixed *claim* sort that has two sub-sorts: *formalClaim* and *informalClaim*. They represent formally or informally (respectively) the evidences for other goals or goal refinements. There is also zero or more parameters associated to every sort. The cost requirements, for example, may have an upper bound parameter.

Link types: The analysis process continues with NFR-goals being decomposed into satisficing goals one or many times. Meanwhile, multiple types of relationships take place between parent goals and one or more of their off-spring goals. These relationships are the link types, and they indicate either a positive or a negative support towards a particular refinement. Links may relate another relationship to an argumentation goal. Thus, link types need to be satisficed as well.

Methods: A goal in this framework can be refined either manually by the designer or using some goal-refinement methods. There are three types of goal-refinement methods in accordance with the previously mentioned goal classes: *goal decomposition methods*, *goal satisficing methods*, and *argumentation methods*. As illustrated earlier, we start with a set of NFR-goals that are refined and expanded until we form an NFR-graph. During the expansion, two lists are maintained: *Open* for the propositions that are to be refined, and *Closed* for propositions that have been completely refined. The process goes as follows. A proposition is selected from the open list to be refined. After that, the designer decides whether the refinement should be done manually or using one the available methods. As the refinement continues, new propositions and links for the new off-springs are created and added to the open list. This process is repeated until no refinements are left and all propositions are placed in the close list.

Labelling procedure: The qualitative labelling procedure assign a label to nodes and links in the constructed graph to determine their satisfaction status. They are labelled *satisficed* (S) if they are satisficeable, *denied* (D) if they are deniable, *conflicting* (C) if it is both, and *undetermined* (U) if it is neither. They are used to qualitatively reason about NFR-frameworks.

1.3 The i^* modelling Framework

The i^* family inherited the concept of soft-goals from the NFR-framework to be able to address them as a first class modelling concept [CdPL09]. As argued in [Yu97], the i^* framework features distinguishing early-phase requirements engineering from late-phase requirements engineering. It is mainly composed of two modelling components: the *Strategic Design (SD)* and the *Strategic Rationale (SR)*. The main concept in the i^* framework is *intentional actors*. An intentional actor is an organizational actor with intentional elements and properties such as *goals*, *tasks*, *resources*, and *soft-goals*. Those actors depend on one another to meet goals and perform tasks. Thus, they can either achieve the impossible together or hinder each other if one of them do not participate adequately.

The *Strategic Dependency* (SD) model basically describes the dependency relationships between intentional actors. An SD model consists of a set of nodes and links connecting the actors. Nodes represent actors and each link represents a dependency between two actors. The depending actor is called *Depender* and the actor who is depended upon is called the *Dependee*. Dependency types are used to differentiate between different kinds of relationships between actors as the SD model does not model relationships between elements or activities. There are four depender-dependee types in SD: goal dependency, resource dependency, soft-goal dependency, and task dependency. A *goal dependancy* is when fulfilling a goal relies on an actor. A *resource dependency* is depending on an actor as a resource, such as an agreement from an actor to provide certain data. A *soft-goal dependency* is a measurement taken with an actor to assure the satisfaction of a soft-goal. Finally, a *task dependency* is reflected through the reliance of a certain task on an actor. The reader can find an illustrative example in [Yu97].

In contrast, the *Strategic Rationale* (SR) model describes the objectives and interests of the stakeholders and how to meet them. While the SD model shows one level of abstraction, the SR model shows a more detailed level of modelling. Also while the SD model expresses external intentional relationships between actors, the SR model expresses internal relationships between the same intentional elements (goals, tasks, resources, and soft-goals) using two types of links: *means-ends links* and *task-decompositions links*. The means-ends links basically answer the *why* question about the reasons to perform certain tasks, whereas the task-decompositions links answer the *how* question about the required actions to achieve those tasks.

The resulted hierarchical graphical representation of the intentional elements and the links among them create a *routine*. A routine is the ability to perform and address a certain objective, and it is granted by knowing the SD and SR models for that objective.

1.4 A Model for Recording the Reasons for Design Decisions

Potts and Bruns [PB88] outlined a model to represent the relationships between *artifacts* and *deliberations*. Artifacts are the system's requirements and specific documentations that give attention to issues regarding the developed system, such as use cases and class diagrams. Nevertheless, deliberations are derived from design artifacts and they can be either justifications or alternatives. An *alternative* is one of many solutions or positions that respond to an issue or an artifact. These alternatives range over the suggestion of creating new artifacts, modifying already existing ones, or not making any design changes at all. On the other hand, a *justification* is a statement that demonstrates the rationale in support of or against the related alternative. One of the main issues during the derivation process is traceability. As in, objects in earlier artifacts must be implemented by objects in later artifacts. Vice versa, objects in later artifacts must be reflected to the earlier artifacts.

In a multi-phase development process, intermediate artifacts are generated to document the design decisions as the derivation continues. However, they only document the results of a previous design phase not a subsequent one. The progression of a single product development results into a graph. This graph has two kinds of design documentations: *nodes* and *links* or *arcs*. Nodes document the design results (artifacts), whereas links document the derivation processes (rationale or alternatives).

1.5 Scenario-Based Analysis of Nonfunctional Requirements

In scenario-based analysis in [SM98], scenario templates are created for nonfunctional requirements and guidelines are provided for scenario creation and validation. Not only do scenarios are used as a method for early exploration of nonfunctional requirements, but they also have the ability to capture the real-world experience and that what makes them suitable and efficient for requirements analysis [Potts et al. 1994].

As nonfunctional requirements interact with each other, we need to trace these interactions and relationships. To investigate relationships between nonfunctional requirements, NFR-scenarios alone are not enough. The following two elements are also required. First, a system model that may be either an architecture, a technical design, a prototype, or an implementation. This system model is required to describe the characteristics of agents, tasks, resources and soft-goals. Second, a design rationale to argue different design options and decisions. Sutcliffe and Minocha [SM98] used the *QOC* notation [Maclean et al. 1991], which expresses design problems as *questions*, functional requirements as *options*, and nonfunctional requirements as *criteria*. A design rationale provides a range of trade-off functional decisions that only have either a positive relationship that help satisfying nonfunctional requirements, or a negative relationship that hinders the satisfaction of nonfunctional requirements. Therefore, scenarios must be run against some views of the system to investigate relationships between nonfunctional requirements, different refined alternatives, and trade-offs to satisfy nonfunctional requirements.

The use of the method in [SM98] starts with the identification and decomposition of nonfunctional requirements into quality criteria to help assess how well the requirements are satisfied. Then, a contextual scenario is created for each and every requirement. Thereafter, scenarios are converted from their narrative-stories format into an executable format that can be executed by the system. Finally, they are assessed against the system model to decide on the most suitable trade-off for the satisfaction of nonfunctional requirements.

Scenario-based analysis is not a modelling technique. It is a way of testing nonfunctional requirements and validating relationships between them.

1.6 Porter and Selby Quality Classification Tree

Nonfunctional requirements are commonly represented by trees expressing the concept of NFR-decomposition [CdPL09]. Another example is the Porter and Selby quality classification tree [PS90]. Porter and Selby's metric-based classification trees are measurement-based models of high-risk components of the system that help identifying and eliminating high-risks whenever they are encountered before they spread and affect the system. The method is called *automatic generation of metric-based classification trees.*

The classification trees can be customized using different kinds of software metrics. Thus, the proposed approach can generate software quality classification trees to distinguish between high-quality modules and low-quality modules. The graphical representation of such tree has two kinds of nodes: a predicate node and a terminal

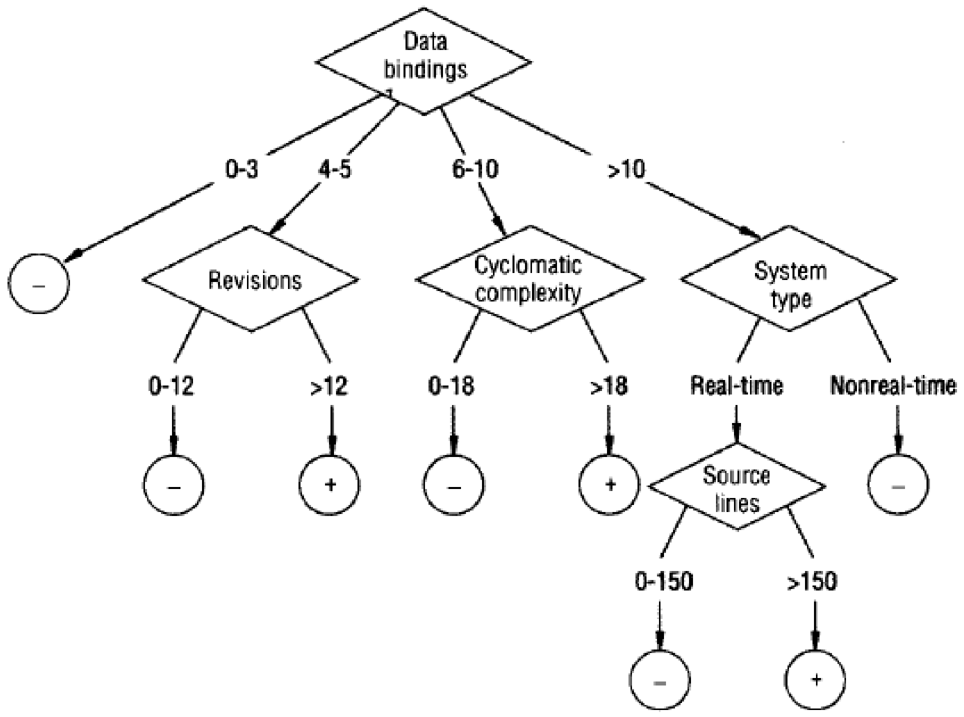


Figure 2.1: A Hypothetical Example of a Quality Classification Tree [PS90]

node. A *predicate node* is for classifying modules into several classes by a certain given factor (software metric). Figure 2.1 shows a hypothetical classification tree inspired by [PS90]. The tree illustrates that only one software metric is used per decision node as a classifying factor. Each category of the classifying metrics corresponds to a tree edge. Each edge is called terminal node, which is a certain quality class. Thus, a *terminal node* is used to determine the quality class of a module that is distinguished from other modules by the predicate node. On a path from a root predicate node to a terminal node there are no duplicate factors.

In general, software quality trees recognize the relationships and interactions between nonfunctional requirements. Whereas in particular, Porter and Selby quality classification tree emphasizes the fact that a predicate quality classifies other quality modules into several terminal classes.

2 Informal Quantitative Modelling Techniques

2.1 Goal-Centric Traceability for Managing NFRs

Traceability is defined as "the ability to describe and follow the life of a requirement in both a forwards and backwards direction" [GF94]. However, tracing nonfunctional requirements is hard because they have an overall impact upon a software system, and also due to the excessive number of interdependencies and trade-offs among nonfunctional requirements [CHSB$^+$05].

As illustrated in [CHSB$^+$05], Goal-Centric Traceability (GCT) equips the developers with the ability to reason about the impact of functional changes upon nonfunctional requirements and therefore maintain the system's quality. This assessment is achieved in a quantitative informal manner. GCT is implemented through the following four phases: *goal modelling*, *impact detection*, *goal analysis*, and *decision making.*

Goal modelling takes place at the early stages of the system's development. During this phase, nonfunctional requirements are modelled as goals and operationalizations within a soft-goal interdependency graph (SIG).

Impact detection phase takes place during the change process. When changes occur to functional requirements, traceability links are dynamically established between the impacted functional elements and the potentially impacted SIG elements. This dynamic approach is definitely due to the excessive number of links and interdependencies between functional and nonfunctional elements. These traces allow developers to identify the direct functional impact of the change and then evaluate its larger impact on SIG elements. A retrieval algorithm is used to identify and return a set of the potential impacted SIG elements. After the user's evaluation and approval, the output of this phase is a set of operationalizations and goals.

During the goal analysis phase, all retrieved goals are re-evaluated. Goal re-evaluation is a recursive process that is applied on each and every operationalization and its parent goal if the satisficing of the parent goal is affected by the proposed change. This continues until all potentially impacted goals have been re-evaluated.The output of this phase is a report identifying all affected goals whether negatively or positively.

Finally, during the decision making phase, stakeholders view the resulted report and decide in support of or against to the implementation of the change. A noteworthy information is that this approach is useful when having two separate models, one for functional requirements and another for nonfunctional requirements. However, since this style of modelling is not of our concern, this approach may as well not concern us.

2.2 KAOS

KAOS [CdPL09] [DvLF93] stands for Knowledge Acquisition in Automated Specification. It is a rigorous goal-oriented modelling technique that was developed in the early 90's. The framework employs the notions of *goals* and *agents* and uses real-time linear temporal logic for defining these entities. For example, both functional and nonfunctional goals are formalized into operators like *Achieve*, *Maintain*, and *Avoid* [CdPL09]. These operators usually operate over past and future states and they semantically capture maximal sets of desired behaviours.

In KAOS, "Goals are generally modelled by intrinsic features such as their type and attributes, and by their links to other goals and to other elements of a requirements model" [vL01]. The approach [DvLF93] starts by defining a set of high-level goals, nonfunctional requirements, as well as a set of agents and actions. Then it iteratively refines these goals and introduces the AND/OR operationalization links to relate the high-level goals to the refined operationalizations. Obstacles and conflicts towards satisfying theses goals are defined later on. The refined operationalizations are software constraints that can be assigned to agents. The approach terminates when all operationalizations are realized by individual agents and can be described as constraints in temporal logic. Thus, KAOS features a great significance of high-level goals since it is a top-down process rather that bottom-up process. It starts from system-level and organizational objectives to reach lower-level descriptions.

3 Formal Quantitative Modelling Techniques

3.1 The Requirement Hierarchy Approach

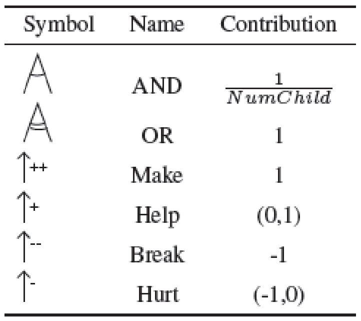
Rayan in [Rya00] proposed a technique called the *Requirement Hierarchy Approach* (RHA). The technique creates a visual tool that helps stakeholders visualize the requirements, their relations, and their affect on the system. Such a tool is very beneficial since stakeholders' requirements tend to have an overlapping nature. Moreover, the Requirements Hierarchy Approach (RHA) can aid in requirements elaboration, validation, and verification. It also helps with the tradeoff analysis since the graphical visual representation eases detection of conflicts.

As put by Ryan [Rya00], nonfunctional requirements can be quantified by the percentage of functional requirements they satisfy. These functional aspects are generated through placing high-level nonfunctional requirements at the top of the hierarchy and gradually decomposing them into functional requirements. The approach uses a weighting procedure to assign weights to top-level NFR requirements based on the experience of stakeholders and domain experts. Using the hierarchy, weights (from 0.1 to 1.0) inspired by stakeholders are assigned to the low-level functional requirements and then traversed upward to calculate a final number for the top NFR-requirement. After the evaluation of the top NFR-requirements, the RHA compares, parses out, and checks for uniqueness among these requirements according to a rigorous classification of nonfunctional requirements. Otherwise, the quantification process will be inconsistent. According to Ryan, the best classification to use is the ISO software quality standard. The ISO/IEC 9126 taxonomy of quality requirements identifies six main internal and external quality characteristics, namely: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. These characteristics are in turn subdivided into sub-characteristics, which is the level where functional requirements are defined and therefore quantification occurs.

In short, the approach helps stakeholders to have a clear vision of the system to be. However, nonfunctional requirements are not always decomposed into functional aspects. Sometimes, they are decomposed into design-decisions, which might complicate their quantification. Moreover, it does not accurately assess nonfunctional requirements quantitatively even though it does require some empirical data as inputs. Therefore, Ryan's quantification method [Rya00] has a quite limited scope of applicability.

3.2 A Process-Oriented Approach Towards Quantitative Reasoning of Nonfunctional Requirements

The work presented by Affleck and Krishna in [AK12] is an extension to the NFR-Framework proposed in [CNYM00]. They want to bridge the gap between nonfunctional requirements and design by providing quantitative support towards the decision-making process. They attempt to calculate the effect of the operationalizations (decomposed nonfunctional requirements) on the developed system. Their proposed extended framework includes several modifications to the original NFR-Framework [AK12]. These modifications are labelled as follows: Original (O), Addition (A), Extension (E), or Replacement (R). The calculation is carried out through



Symbol	Name	Contribution
	AND	$\frac{1}{NumChild}$
	OR	1
$\uparrow^{++}$	Make	1
$\uparrow^{+}$	Help	(0,1)
$\uparrow^{--}$	Break	-1
$\uparrow^{-}$	Hurt	(-1,0)

Figure 2.2: Weighting of Contribution Relationships in SIGs [AK12]

the following steps:

- Steps 1&2: Identify Soft-goals (O) & Decompose Soft-goals (E): Similar to the original framework, nonfunctional requirements are identified and decomposed. However, the decomposition process is extended to assign weights to the contribution of the decomposed children towards the satisfaction of their parents. The weighting procedure is defined according to the table in figure 2.2.

- Step 3: Assign Leaf-Soft-goal Weights (A): Weights are assigned to the leaf-soft-goals. The weighting is based on the interval unit of $[0, 1]$ with 0 being non-critical and 1 being most critical. This weight is referred to as LSG_{weight}.

- Step 4: Identify Operationalizations (E): The original framework [CNYM00] identify operationalizations yet this step is extended in [AK12] to assign weights to the relationships between leaf-soft-goals and operationalizations according to the table in figure 2.2 as well. The relationship is referred to by $impact_{LSG \times OP}$.

- Step 5: Calculate Operationalization Scores (A): The following equation is applied in a top-down manner to calculate the individual score of operationalizations.

  $$OP_{score} = OP_{parent.score} + \sum_{LSG} impact_{LSG \times OP} \times LSG_{weight}$$

  Where OP_{score} is the quantified affect of the operationalization upon the system in case it was implemented, and $OP_{parent.score}$ is the score of the parent operationalization. In case of having no parent, a score of 0 is assigned to $OP_{parent.score}$.

- Step 6: Select Operationalizations (R): The OP_{score} clarifies whether the operationalization is accepted or rejected. The score varies among three ranges: $(0, \infty)$ for positive affect, $(-\infty, 0)$ for negative affect, or 0 for no affect. Operationalizations that directly impact leaf-soft-goal are accepted if their score is positive or equal to zero. Otherwise, they are rejected. Operationalizations of an AND contribution are also accepted when their score is greater than or equal to zero, and rejected otherwise. However, the rejection of an AND operationalization indicates the rejection of its accepted parent due to a conflict. The involvement of the software developer is then required. Finally, operationalizations of an OR contribution are accepted under two conditions. First, they must have either a positive affect or no affect. Second, their benefit to the system has to be greater than their parent.

- Step 7: Calculate Leaf-Soft-goal Scores (R): The satisfaction of the leaf-soft-goal is generated by adding up the scores of the accepted operationalizations. The overall score is contained within $[-1.0, 1.0]$ using the following equation:

  $$LSG_{score} = max(min(\textstyle\sum_{OP_{accept}=true} impact_{LSG \times OP}, 1), -1)$$

- Step 8: Calculate Soft-goal Scores (R): The score of a soft-goal is calculated via the equation:

  $$SG_{score} = max(min(\textstyle\sum SG_{child} \times SG_{child.contrib}, 1), -1)$$

  Where SG_{child} is the score of a child soft-goal, and $SG_{child.contrib}$ is the percentage by which a child soft-goal contribute to satisfice its parent soft-goal.

- Step 9: Calculate Attainment (A): Both an actual and optimal attainments can be calculated. An optimal attainment is when a nonfunctional requirement has been fully satisficed. This is done through:

  $$attainment_{optimal} = \textstyle\sum_{LSG} LSG_{weight}$$

  Nevertheless, an actual attainment is calculated according to the percentage each leaf-soft-goal is attained:

  $$attainment_{actual} = \textstyle\sum_{LSG} max(LSG_{score} \times LSG_{weight}, 0)$$

An illustrative bank system example can be found in [AK12]. An evaluation of both the original framework and the extension showed no difference in decisions. In fact, it was discovered that the proposed extension provided more detailed feedback then its qualitative alternative.

4 Modelling of Nonfunctional Requirements in Product Families

In this section, we talk about the basis of the product family concept in software engineering. Then, we elaborate on a formal algebraic technique used for analyzing and constructing product families mathematically. The content and examples of this section is mainly taken from [HKM11].

4.1 The Basis of Product Families

A product family is a set of products with common basic features. Features are variants of requirements and components. According to [HKM11], “a feature is a conceptual characteristic visible to stakeholders“. However, at the requirements level and according to Savolainen et al. [SOMZ05], “a feature is specified by a set of requirements; this set may contain one or more requirements“. An example taken from [BLP05] describes a company that has three lines of products: MP3 players, DVD players, and Hard Disk recorders. Members of this family have some *commonalities*, such as having an audio equalizor. They also have *mandatory* features like the ability to play mp3 files for a MP3 player, and other *optional* features such as the ability to record mp3 files for MP3 players.

There are various ways to analyze and model product families or software families. One widely-known and widely-used approach is called feature modelling. *Feature modelling (FM)* is an approach to analyze, build, and represent compact software families on the basis of their features. A *feature*, in this context, is a distinctive quality attribute or a user-defined aspect of the software system to be. A *feature model* defines features and their derivatives and dependencies. Feature models are the resulted reference architecture and they are widely used during the software development process.

There are many feature modelling techniques. Some of these techniques are graphical, which result into graphical representations of software families called *feature model diagrams*. The other some are algebraic non-graphical techniques called *product family algebra*. All of these techniques share the same concept of features and feature models.

An example of feature model diagrams is FODA. FODA stands for *Feature Oriented Domain Analysis* and it was the first to introduce feature models. A graphical feature model in FODA is composed of nodes. Each *node* represents a feature. The *root node* represents the main concept of the model. A node can be either external, which is a *primitive* feature represented by a leaf on the edge, or internal, which is a *compound* feature of other smaller subsequent features. In case of compound features, underneath features can be either composite using the *AND* notation or alternated using the *XOR* notation. Finally, a feature can be *mandatory* or *optional*. A mandatory feature is a feature that must be addressed in the product, while an optional feature is a feature that can be addressed in the product.

4.2 Product Family Algebra (PFA)

Product family algebra is a mathematical calculus-oriented way of representing product families. It can capture commonality and variability of product families and allow manipulations upon product families in a mathematical way.

Definition 4.1 (Semiring [HKM11])**.** *A semiring is a quintuple* $(S, +, 0, \cdot, 1)$ *such that* $(S, +, 0)$ *is a commutative monoid and* $(S, \cdot, 1)$ *is a monoid such that* $\cdot$ *distributes over* $+$ *and* 0 *is an annihilator, i.e.,* $0 \cdot a = 0 = a \cdot 0$*. The semiring is commutative if* $\cdot$ *is commutative and it is idempotent if* $+$ *is idempotent, i.e.,* $a + a = a$*. In the latter case, the relation* $a \leq b \overset{def}{=} a+b = b$ *is a partial order (i.e., a reflexive, antisymmetric and transitive relation) called the natural order on* S*. It has* 0 *as its least element. Moreover,* $+$ *and* $\cdot$ *are isotone with respect to* $\leq$*.* ■

Product family algebra is an idempotent commutative semiring with the carrier set S being the set of all product families. The binary operator $\cdot$ is interpreted as the composition of product families and is used to represent mandatory features. Whereas the binary operator $+$ is interpreted as the choice between product families and is used to represent optional features. 0 is the empty product family, and 1 is a product with no feature.

A product family is a set of products. A family that is indivisible with regard to $+$ is a product.

With the two operators $+$ and $\cdot$, product family algebra can express the relationships between nonfunctional requirements in different models.

Product [HKM11]: A product family a is a product if:

- $\forall(b \mid: b \leq a \implies b = 0 \vee b = a)$ and
- $\forall(b, c \mid: a \leq b + c \implies a \leq b \vee a \leq c)$.

In particular, 0 is a product. A product a is proper if $a \neq 0$.

Feature [HKM11]: An element a is called a feature if it is a proper product and:

- $\forall(b \mid: b | a \implies b = 1 \vee b = a)$ and
- $\forall(b, c \mid: a | (b \cdot c) \implies a | b \vee a | c)$

where the divisibility relation $|$ is given by $x | y \iff \exists z : y = x \cdot z$

Based on the natural ordering relation provided with a semiring structure, we define a notion of refinement that is used later on to define other relations among families.

The following are expressions of some graphical feature modelling concepts into PFA terms. The AND- composition of features b, c, and d for a product a is expressed with the sentence $a = b \cdot c \cdot d$. The XOR- decomposition of features b, c, and d for a product a is expressed with the sentence $a = b + c + d$. However, an optional feature b and a mandatory feature c of some product a is expressed with the sentence $a = (b + 1) \cdot c$.

See [HKM11] for further discussion regarding Product Family Algebra (PFA).

4.3 Model-Based Verification of Quantitative Nonfunctional Requirements in Product Lines

This framework is an extension to a framework proposed in [GS11], which was designed to assess nonfunctional requirements in single products or systems. The extended framework however is used to verify nonfunctional requirements in software product lines. It derives Markov models and then assess these models against nonfunctional requirements using probabilistic model checkers [GS13]. Probabilistic Markov models are basically state machines in which transitions are labelled by probabilities. Nevertheless, the Markov models presented here substitute probabilities on transitions with variables called *parameters*, and therefore the resulted models are referred to by *parametric Markov models.*

The proposed framework [GS13] verifies nonfunctional requirements in Software product line (SPL) in two approaches, a product-by-product approach and a parametric approach. The *product-by-product* fashion instantiates a behavioural model for every product in the product line, transforms it into a Markov model, and then verifies it against nonfunctional requirements. This approach is only feasible with a small number of products due to the expensive verification time consumption. However, in the case of large number of products, we use the *parametric approach.* After splitting the products into common and variant parts, the parametric approach derives behavioural models out of both common and variant parts among products, transforms them into Markov models, and verifies them. Thereafter, the results are composed to provide evaluation. Accordingly, each property in the product line is analyzed only once.

The modelling process [GS13] is based on the use of *UML Sequence Diagrams (UML SDs)*. However, in order to capture the variability, they are in this framework augmented with variation points and alternatives that are linked to the the feature model of the product line. Thus, they are called *variable UML SDs*. These diagrams support messages, lifelines, and fragments.

In conclusion, most existing NFR-quantification techniques promote the use of feature as a main artifact [MA09]. However, this approach focuses on the use of behavioural models. They show variation on actions and not on features. Besides, it deals with non-determinism more than quantification. Since we are attempting to quantify features and feature-interactions, this model seems irrelevant to our research.

5 Conclusion

A major drawback of all previously mentioned techniques is that they model non-functional requirements on the basis of the overall former experience of the software architect. Not to mention that the assessment is mostly qualitative. Another drawback of these techniques is that some of them are graphical representations, which are very limited in use when expanding the system model. Also, apart from the product family algebra, the rest of the listed models can only model nonfunctional requirements in single products and cannot be used in product families. This contradicts with the interests of many organizations that adopt the concept of software product families.

Some quantitative models were presented as well. However, as far as we surveyed the literature, none of the quantitative models seemed to be quiet efficient. For instance, the Requirement Hierarchy Approach (RHA) is limited only to the quality requirements that are decomposed into functional requirements. Moreover, the framework proposed by Affleck and Krishna for is quiet relevant to what we are trying to accomplish. However, the values of the contribution relationships are insufficient to us since we would like to provide the flexibility of using a wide range of values to express the nature of the contribution. Nevertheless, the model-based framework quantifies actions and not features, and addresses non-determinism rather than quantification.

The question here is why not to have a quantitative evaluation for the satisfaction of nonfunctional requirements in a certain system? This quantitative evaluation can be founded on the partial quantitative evaluation of each and every system component to these nonfunctional requirements. This evaluation can as well be carried out using a function that illustrates the effect of these system components and their evaluation on each other.

Moreover, the challenges of quantifying nonfunctional requirements in product families concludes the need to develop a model that specifically addresses that issue. The model-to-be should at least include the following specifications. The model should handle nonfunctional requirements in an efficient manner. Inputs to the system model are empirical data resulted from feature interactions between system components. Most importantly, this model has to be founded on an a formal algebraic structure, which enables us to analyze nonfunctional requirements and draw calculations to assess the satisfaction of each nonfunctional requirement by each component in the system or by each product in the product family.

Developing such a model is mainly the objective of this research. The next two chapters will present this model in details.

Chapter 3

Mathematical Backgrounds

In this chapter, we introduce the necessary mathematical general concepts required for the understanding of the material presented in the remaining part of this thesis. In Section 3.1, we give definitions of some properties of binary operations. In Section 3.2, we define needed algebraic structures: *semigroups*, *monoids*, *semirings*, and Γ-*semirings*. Finally, preordered sets and the measurement scale of intervals are tackled in Section 3.3.

1 Properties of Binary Operations

The term *algebraic structure* refers to some arbitrary set called *carrier set* and a set of operations defined on the set. Given a carrier set S and binary operators (i.e., $+$ and $\cdot$) defined on S, we have the following basic operations:

Closure: The set S is said to be *closed* under $\cdot$ if and only if $\forall(a, b \mid a, b \in S : a \cdot b \in S\,)$.

Commutativity: The operator $\cdot$ is said to be *commutative* if and only if $\forall(a, b \mid a, b \in S : a \cdot b = b \cdot a)$.

Associativity: The operator $\cdot$ is said to be *associative* if and only if $\forall(a, b, c \mid a, b, c \in S : (a \cdot b) \cdot c = a \cdot (b \cdot c))$.

Idempotence: The operator $\cdot$ is said to be *idempotent* if and only if $\forall(a \mid a \in S : a \cdot a = a)$.

Left Distributivity: We say that the operator $\cdot$ is *left distributable* over the operator $+$ if and only if $\forall(a, b, c \mid a, b, c \in S : a \cdot (b + c) = a \cdot b + a \cdot c)$.

Right Distributivity: We say that the operator $\cdot$ is *right distributable* over the operator $+$ if and only if $\forall(a, b, c \mid a, b, c \in S : (a + b) \cdot c = a \cdot c + b \cdot c)$.

Identity: An element $e \in S$ is said to be the *identity element* if and only if $\forall(a \mid a \in S : a \cdot e = e \cdot a = a)$. Such an element is unique and neutral.

Annihilation: An element $0 \in S$ is said to be the *annihilator* if and only if $\forall(a \mid a \in S : a \cdot 0 = 0 \cdot a = 0)$.

2 Algebraic Structures

Definition 2.1 (Semigroup). *A semigroup is an algebraic structure $(S, \cdot)$ where S is a nonempty carrier set closed under the operator $\cdot$, and $\cdot$ is associative.* ■

One of the most familiar examples of a semigroup is the set of positive integers $\mathbb{Z}^+$ with the addition $(\mathbb{Z}^+, +)$.

Definition 2.2 (Monoid). *A monoid is a semigroup that has an identity.* ■

Definition 2.3 (Semiring). *A semiring (or a hemiring or a near-ring) algebraic structure is a quintuple $(S, +, 0, \cdot, 1)$ consisting of a nonempty carrier set S equipped with two binary operations, multiplication $\cdot$ and addition $+$, such that:*

1. *$(S, +)$ is a commutative monoid.*
2. *$(S, \cdot)$ is a monoid.*
3. *Multiplication left and right distributes over addition.*
4. *Multiplication has 0 as an annihilator.*

■

Therefore, a semiring satisfies the following axioms: closure, associativity, identity, commutativity, left distributivity, right distributivity, and has an annihilator.

Definition 2.4 (Γ-Semiring)**.** *The triple $\big(S, \Gamma, +\big)$ where S and Γ are disjoint is called a Γ-semiring if we have the mapping $S \times \Gamma \times S \to S$, where $\big(S, +\big)$ and $\big(\Gamma, +\big)$ are additive commutative monoids, and if it satisfies the following for every a, b, c in S and for every α, β, γ in Γ:*

1. $a\alpha(b\beta c) = (a\alpha b)\beta c$
2. $a\alpha(b + c) = a\alpha b + a\alpha c$
3. $(a + b)\alpha c = a\alpha c + b\alpha c$
4. $a(\alpha + \beta)b = a\alpha b + a\beta b$

■

An *additive* monoid is a monoid equipped with the binary operation addition +, such as the two monoids $\big(S, +\big)$ and $\big(\Gamma, +\big)$ of a Γ-semiring. According to Definition 2.4, a Γ-semiring satisfies the following axioms: closure, associativity, identity, commutativity, and the four axioms of Definition 2.4.

The Γ-semiring $\big(S, \Gamma, +\big)$ is commutative if every α in Γ is commutative (i.e., $a\alpha b = b\alpha a$), and it is idempotent if the addition is idempotent (i.e., $a + a = a$).

3 Intervals

Definition 3.1 (Preordered Set). *A preorder is a binary relation $\leq$ defined over a set P, which satisfies the following for all $a, b, c \in P$:*

1. *Reflexivity: $a \leq a$*
2. *Transitivity: $a \leq b \wedge b \leq c \implies a \leq c$*

A set equipped with a preorder relation is called a preordered set. ■

Definition 3.2 (Intervals). *Let $(P, \leq)$ be a preordered set. Let $a, b \in P$ with $a \leq b$. Then we have an interval I such that:*

$$I = [a, b] \overset{def}{=} \{x \in P \,|\, a \leq x \leq b\}$$

■

The elements a and b of the interval $[a, b]$ are called *endpoints*. An *open* endpoint is an excluded endpoint from the interval, and it is denoted by parentheses or reversed square brackets. A *closed* endpoint is an included endpoint in the interval, and it is denoted by regular square brackets.

An interval that includes both of its endpoints is called a *closed* interval. Whereas, an interval that excludes both of its endpoints is called a *open* interval. A *degenerate* interval is a set that includes a single element only. An *empty* interval is an interval that has no element whatsoever.

4 Conclusion

In this chapter, we reviewed some basic mathematical concepts. We started with some properties for binary operations such as: *closure*, *commutativity*, *associativity*, *idempotence*, *right distributivity*, *left distributivity*, *identity*, and having an *annihilator*. After that, we covered some algebraic structures such as *semigroups*, *monoids*, *semirings*, and Γ-*semirings*. A semigroup satisfies two axioms, closure and associativity. A monoid is basically a semigroup with an identity axiom. A semiring is a structure composed of two additive commutative monoids that have an annihilator and satisfy the right distributivity and left distributivity axioms alongside the axioms of monoids. A Γ-semiring is a semiring with four more axioms listed in Definition 2.4. Finally, we defined *preordered sets* and *intervals*. All these concepts will be put in good use towards the construction of our proposed algebraic technique in Chapter 4.

Chapter 4

Quantitative Assessment of Nonfunctional Requirements in Product Families

In this chapter, we present a formal framework for the quantitative assessment of nonfunctional requirements in product families. In Section 4.1, we present the proposed algebraic system.

1 The Proposed Approach

Quantifying nonfunctional requirements, NFR-relationships, and NFR-assessment is feasible in many cases of quantifiable nonfunctional requirements. For example, defining the performance attribute of an online media broadcast server [TFQ06] where many factors can be defined and evaluated quantitatively (i.e., bandwidth and throughput).

In product families, due to the concept of commonality and variability, families can be generated from a set of features in variant alternate ways. While integrating different families, or in case of composite product family, nonfunctional requirements of that family can be affected. For example, given a product family that has two nonfunctional requirements (e.g., performance and security) we can build families from features that their interactions could increase the performance but sacrifice security, or vice versa. In addition, nonfunctional requirements affect the satisfaction of one another according to several relationships. These relationships are mostly qualitative and can improve or hinder the satisfaction of nonfunctional requirements on many levels.

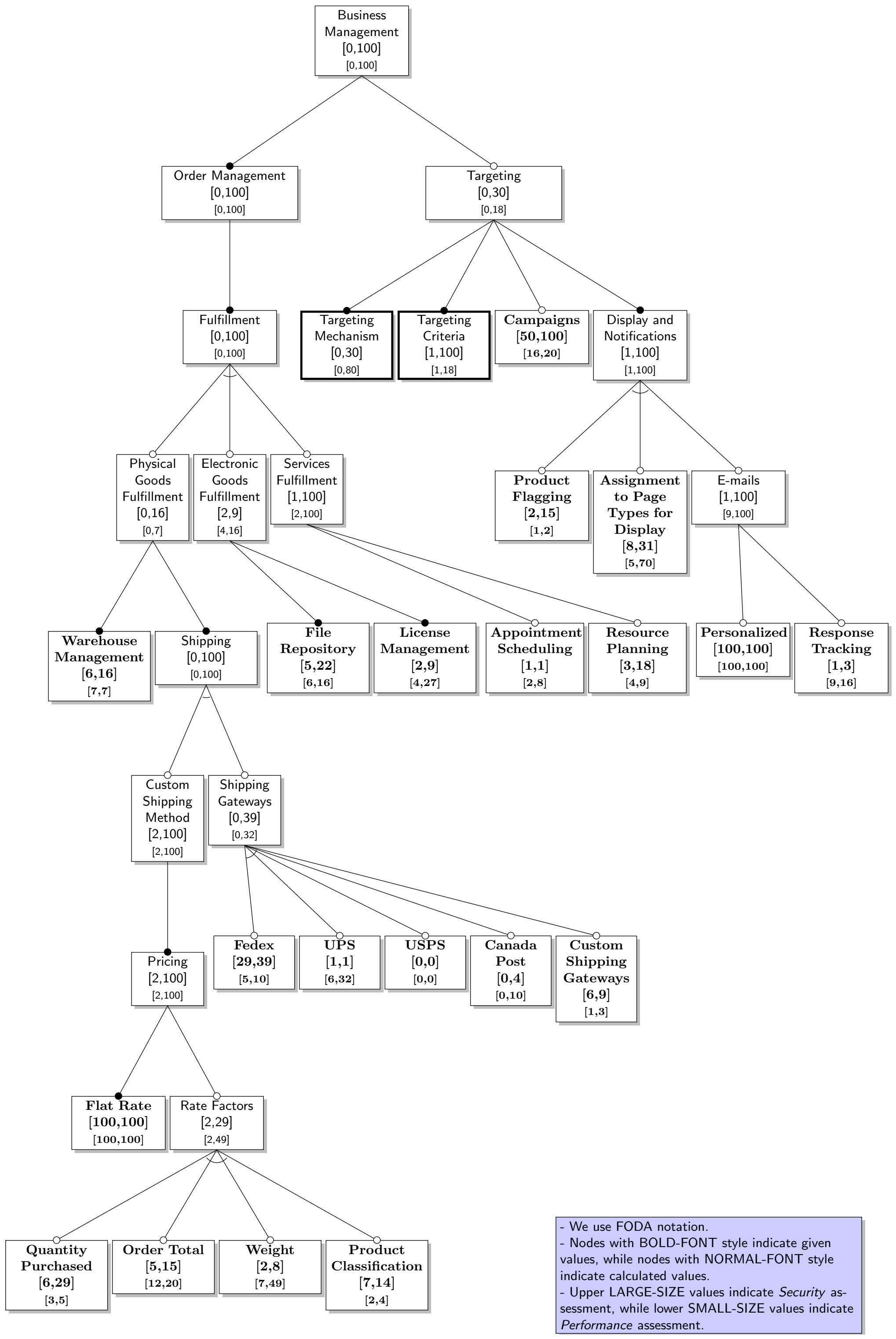
Instead of focusing on only one way of feature-interaction while building product families from a given set of features, we intend through this proposed approach to investigate numerous ways of feature interactions. At the same time, we aim to calculate the quantitative values assigned to the satisfaction of the considered and quantifiable nonfunctional requirements of the product family. These goals can be captured through the use of the algebraic system of Γ-semirings and empirical data. Thereafter, when given a Γ-semiring's term like $x\alpha y$, such that $\alpha \in \Gamma$, we interpret it as follows: "The family formed by the mandatory composition of sub-families x and y, while considering the nonfunctional requirement α“.

For a better understanding, we provide an example to help us walk through the proposed approach. In Figures 4.1, 4.2, and 4.3, we illustrate a feature model inspired by the *Online-Store* example in [Lau06]. However, our feature model only features the *Business-Management* part of the original *Online-Store*.

As shown in Figures 4.1, 4.2, and 4.3, it is quite difficult to deal with nonfunctional requirements graphically due to time and space consumption. The diagrams become very quickly cluttered as soon as we reach a family with a moderate size. However, we instead provide a way to handle nonfunctional requirements textually and systematically, which is more convenient.

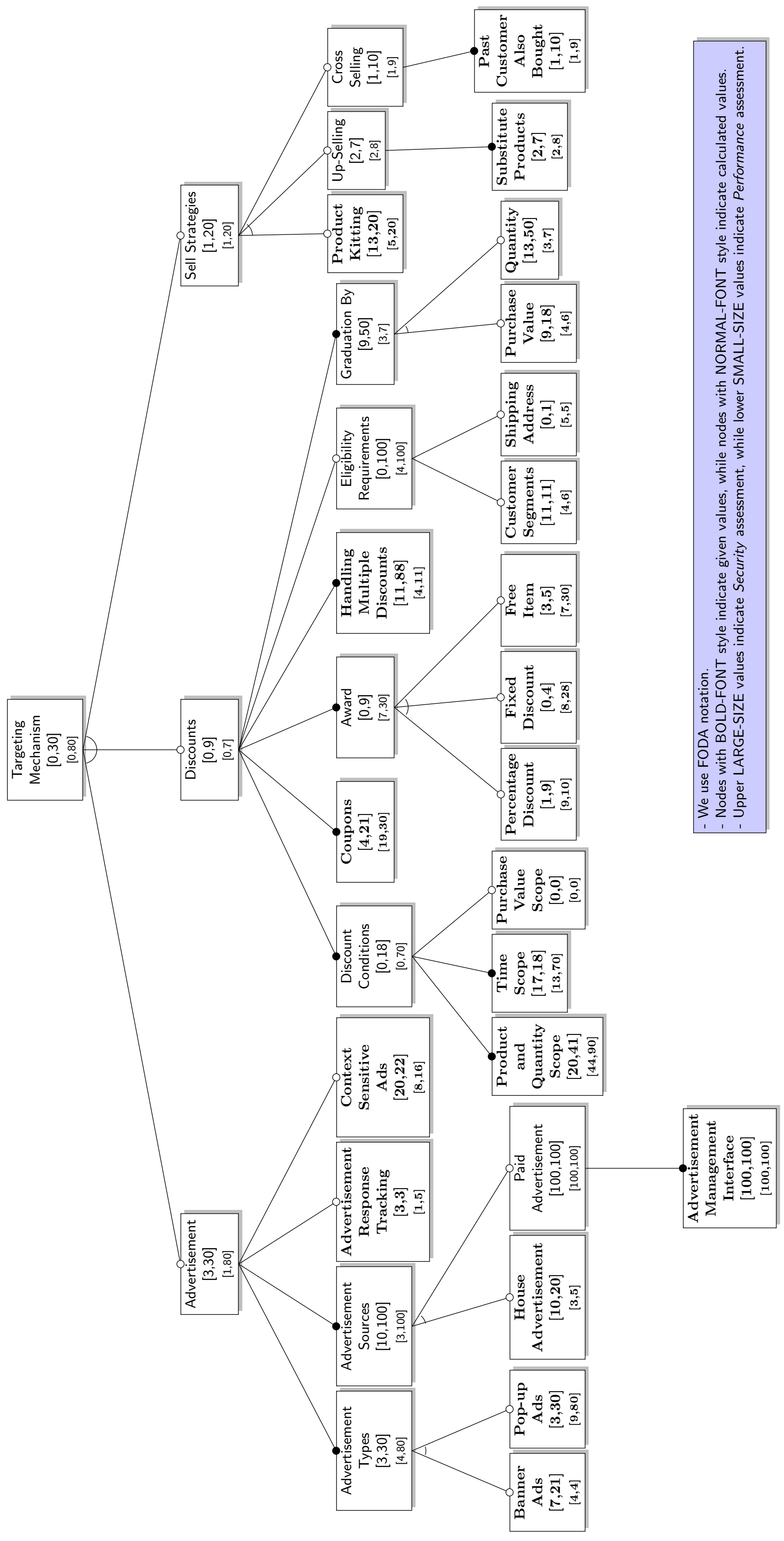
1.1 Intervals to Measure The Weight of Nonfunctional Requirements

The form of the quantitative values we assign to the satisfaction of nonfunctional requirements is one of the prominent features of our algebraic system. When measuring the weight of nonfunctional requirement, it is hard to have an exact singular value for that measure. Besides, it is a bit unrealistic since we do not have (up until now) such precise quantifying techniques for assessing nonfunctional requirements. We will most likely get a range of values, which starts with a minimum potential value and ends with a maximum potential value. For that we use the concept of intervals.


Business Management [0,100] [0,100]
Order Management [0,100] [0,100]
Targeting [0,30] [0,18]
Fulfillment [0,100] [0,100]
Targeting Mechanism [0,30] [0,80]
Targeting Criteria [1,100] [1,18]
Campaigns [50,100] [16,20]
Display and Notifications [1,100] [1,100]
Physical Goods Fulfillment [0,16] [0,7]
Electronic Goods Fulfillment [2,9] [4,16]
Services Fulfillment [1,100] [2,100]
Product Flagging [2,15] [1,2]
Assignment to Page Types for Display [8,31] [5,70]
E-mails [1,100] [9,100]
Warehouse Management [6,16] [7,7]
Shipping [0,100] [0,100]
File Repository [5,22] [6,16]
License Management [2,9] [4,27]
Appointment Scheduling [1,1] [2,8]
Resource Planning [3,18] [4,9]
Personalized [100,100] [100,100]
Response Tracking [1,3] [9,16]
Custom Shipping Method [2,100] [2,100]
Shipping Gateways [0,39] [0,32]
Pricing [2,100] [2,100]
Fedex [29,39] [5,10]
UPS [1,1] [6,32]
USPS [0,0] [0,0]
Canada Post [0,4] [0,10]
Custom Shipping Gateways [6,9] [1,3]
Flat Rate [100,100] [100,100]
Rate Factors [2,29] [2,49]
Quantity Purchased [6,29] [3,5]
Order Total [5,15] [12,20]
Weight [2,8] [7,49]
Product Classification [7,14] [2,4]

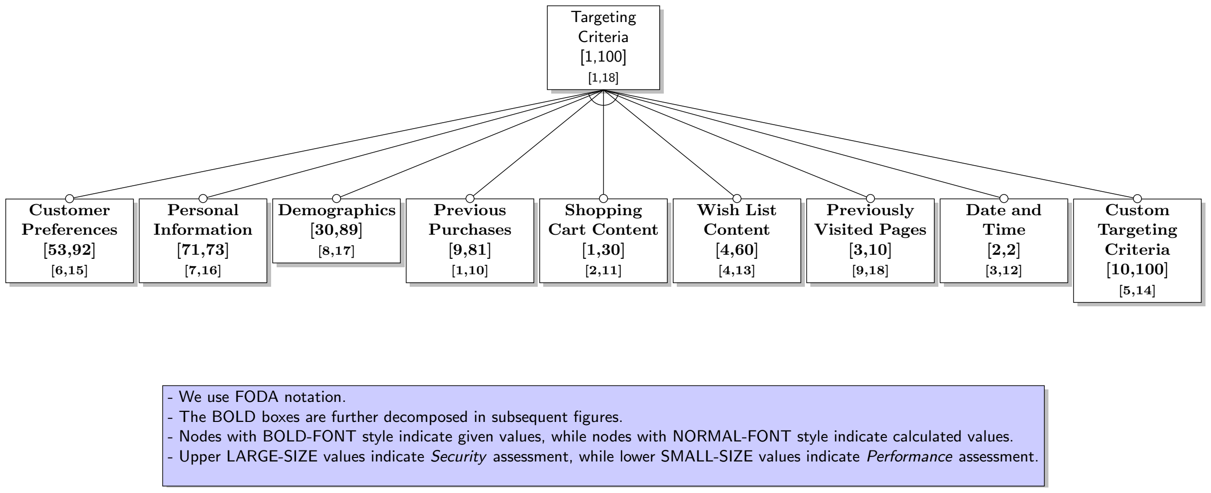
- We use FODA notation.
- Nodes with BOLD-FONT style indicate given values, while nodes with NORMAL-FONT style indicate calculated values.
- Upper LARGE-SIZE values indicate *Security* assessment, while lower SMALL-SIZE values indicate *Performance* assessment.

Figure 4.1: The Business Management Feature Model 1 − 1 (inspired by [Lau06])


Targeting Mechanism [0,30] [0,80]
Advertisement [3,30] [1,80]
Discounts [0,9] [0,7]
Sell Strategies [1,20] [1,20]
Advertisement Types [3,30] [4,80]
Advertisement Sources [10,100] [3,100]
Advertisement Response Tracking [3,3] [1,5]
Context Sensitive Ads [20,22] [8,16]
Discount Conditions [0,18] [0,70]
Coupons [4,21] [19,30]
Award [0,9] [7,30]
Handling Multiple Discounts [11,88] [4,11]
Eligibility Requirements [0,100] [4,100]
Graduation By [9,50] [3,7]
Product Kitting [13,20] [5,20]
Up-Selling [2,7] [2,8]
Cross Selling [1,10] [1,9]
Banner Ads [7,21] [4,4]
Pop-up Ads [3,30] [9,80]
House Advertisement [10,20] [3,5]
Paid Advertisement [100,100] [100,100]
Advertisement Management Interface [100,100] [100,100]
Product and Quantity Scope [20,41] [44,90]
Time Scope [17,18] [13,70]
Purchase Value Scope [0,0] [0,0]
Percentage Discount [1,9] [9,10]
Fixed Discount [0,4] [8,28]
Free Item [3,5] [7,30]
Customer Segments [11,11] [4,6]
Shipping Address [0,1] [5,5]
Purchase Value [9,18] [4,6]
Quantity [13,50] [3,7]
Substitute Products [2,7] [2,8]
Past Customer Also Bought [1,10] [1,9]

- We use FODA notation.
- Nodes with BOLD-FONT style indicate given values, while nodes with NORMAL-FONT style indicate calculated values.
- Upper LARGE-SIZE values indicate *Security* assessment, while lower SMALL-SIZE values indicate *Performance* assessment.

Figure 4.2: The Business Management Feature Model 1 – 2 (inspired by [Lau06])


Targeting Criteria [1,100] [1,18]
Customer Preferences [53,92] [6,15]
Personal Information [71,73] [7,16]
Demographics [30,89] [8,17]
Previous Purchases [9,81] [1,10]
Shopping Cart Content [1,30] [2,11]
Wish List Content [4,60] [4,13]
Previously Visited Pages [3,10] [9,18]
Date and Time [2,2] [3,12]
Custom Targeting Criteria [10,100] [5,14]

- We use FODA notation.
- The BOLD boxes are further decomposed in subsequent figures.
- Nodes with BOLD-FONT style indicate given values, while nodes with NORMAL-FONT style indicate calculated values.
- Upper LARGE-SIZE values indicate *Security* assessment, while lower SMALL-SIZE values indicate *Performance* assessment.

Figure 4.3: The Business Management Feature Model $1-3$ (inspired by [Lau06])

Let I be a set of intervals within $[0,...,N]$ for some N in $\mathbb{R}^+$. Let $I \triangleq \{[a,b] |\ a \leq b \leq N \wedge a,b \in \mathbb{R}^+\}$. We have $(I, \leq_I)$ is a preorder on I, where for $[a_1,b_1]$, $[a_2,b_2] \in I$, we have:

$$[a_1,b_1] \leq_I [a_2,b_2] \quad \overset{\text{def}}{\iff} \quad b_1 \leq b_2$$

The set $(I, \leq_I)$ satisfies the following for all intervals in I:

1. reflexivity due to the reflexivity of $\leq$ on $\mathbb{R}^+$.

2. transitivity due to the transitivity of $\leq$ on $\mathbb{R}^+$.

Therefore, $(I, \leq_I)$ is a pre-ordered set or quasi-ordered set. For detailed proofs, we refer the reader to the Appendix.

According to what we observe, nonfunctional requirements can enhance the satisfaction of each other on many levels, and we need the ability and flexibility to express that. Choosing N as an upper bound can help us achieve that. This choice is as arbitrary as choosing 0 to be the lower bound for intervals. Both endpoints of $[0,...,N]$ are simple choices. One can take $[n, ..., m]$ for $n, m \in \mathbb{R}^+$. In dealing with our *Business-Management* case study in Figures 4.1, 4.2, 4.3, we take $N = 100$.

In decision making, when the goal cannot be satisfied absolutely and the optimal solution cannot be reached, the notion of satisficing tends to select the solution that seems to address most needs. We use the notion of satisficing with our defined pre-order relation $\leq_I$. Let i_1 and i_2 be two intervals such that $i_1 \leq_I i_2$. Thus, we say that i_2 (respectively, i_1) is more (respectively, less) satisficing of the non functional requirement.

We exemplify in Figures 4.1, 4.2, and 4.3 an assessment of the security and performance nonfunctional requirements for the *Business-Management* feature model inspired by [Lau06]. The graphs show the use of intervals to represent the assessment values.

For every $[a_1, b_1], [a_2, b_2]$ in I, we define two operations $\oplus$ and $\odot$, such that:

1. $[a_1, b_1] \oplus [a_2, b_2] \;=\; [min\ (a_1, a_2), max\ (b_1, b_2)]$

2. $[a_1, b_1] \odot [a_2, b_2] \;=\; [min\ (a_1, a_2), min\ (b_1, b_2)]$

For instance, let us have two intervals $[5, 13]$ and $[2, 10]$. We have:
$[5, 13] \oplus [2, 10] \;=\; [min\ (5, 2), max\ (13, 10)] \;=\; [2, 13]$
$[5, 13] \odot [2, 10] \;=\; [min\ (5, 2), min\ (13, 10)] \;=\; [2, 10]$

Since 0 is the least possible value and N is the highest possible value, we conclude that $[0, N]$ is an absorbent element for $\oplus$. Likewise, $[N, N]$ acts as a neutral element for $\odot$, and $[0,0]$ is the absorbent element for $\odot$.

Let us have an interval $[5,13]$ to test and validate these conclusions against it:

$[5,13] \oplus [0,N] \ = \ [min\ (5,0), max\ (13,N)] \ = \ [0,N]$

$[5,13] \odot [N,N] \ = \ [min\ (5,N), min\ (13,N)] \ = \ [5,13]$

$[5,13] \odot [0,0] \ = \ [min\ (5,0), min\ (13,0)] \ = \ [0,0]$

The operations on I, $\oplus$ and $\odot$, are *commutative*, *associative*, and *idempotent* due to the properties of the *min* and *max* functions.

Lemma 1.1. *I is closed with respect to $\oplus$ and $\odot$.*

To prove that I is closed with respect to $\oplus$, we need to prove that $\forall(i_1, i_2 \mid i_1, i_2 \in I \ : \ i_1 \oplus i_2 \in I\,)$. Also, to prove that I is closed with respect to $\odot$, we need to prove that $\forall(i_1, i_2 \mid i_1, i_2 \in I \ : \ i_1 \odot i_2 \in I\,)$. For detailed proofs, we refer the reader to the Appendix.

1.2 Quantitative Assessment Function μ

Let $(S, \Gamma, +, 0_s, 1_s)$ be a mathematical structure where N_f is the set of given non-decomposable nonfunctional requirements to be considered in the system, and Γ is the power set of N_f that we denote by $\mathcal{P}(N_f)$. For example, if we have two nonfunctional requirements to be considered such as security and performance, then we take $N_f \overset{\text{def}}{=} \{s,p\}$ and therefore $\Gamma = \{\emptyset, \{s\}, \{p\}, \{s,p\}\}$.

Let S be a set of ordered tuples on Γ such that an element in Γ-semiring S takes the form of $(\mathcal{P}(\mathbb{F}) \times \Gamma)^{|\Gamma|}$.

To illustrate an interpretation of a term in this structure, we adopt *Kuratowski's* representation of ordered pairs [Kur21], where a tuple (a, b) is represented by:

$$(a, b) = \{\{a\}, \{a, b\}\} \subseteq \mathcal{P}(A \cup B)$$

Therefore, 0_s is $\{\}$. For an element, $\alpha \in \Gamma$, we define:

$$\emptyset_\alpha = \{\emptyset, \pi_{\beta \subseteq \alpha}(\{\emptyset\}, \beta)\}$$

where π is the Cartesian Product.

With Kuratowski's representation, the operation $+$ is translated into set union. Hence, $(S, \Gamma, +, 0_s, 1_s)$ is a commutative idempotent Γ-semiring and $+$ is set union on the Kuratowski's representation of ordered pairs or on set of elementary non-decomposable nonfunctional requirements. We call this interpretation of Γ-semiring, *The Ordered Pairs Model.*

Let $\big(S, \Gamma, +, 0_s, 1_s\big)$ be the ordered pairs model of a commutative idempotent Γ-semiring where S is freely generated from a set of basic features $\mathbb{F}$. For an $\alpha \in \Gamma$ and a set of intervals I provided by stakeholders, we define the function:

$$\mu_\alpha : S \to I$$

The function μ_α is responsible for mapping all families in S to weight values in the form of intervals I.

Note: If we do not consider any nonfunctional requirements (i.e., $N_f = \emptyset$), then we have a Γ-semiring $\big(S, \{\emptyset\}, +, 0_s, 1_s\big)$. Since we only have one operator $\{\emptyset\}$ in Γ, then we can denote that operator by the $\cdot$ operator. Therefore, we have an isomorphic semiring to $\big(S, \cdot, +, 0_s, 1_s\big)$. The latter semiring forms a product family algebra when $+$ is idempotent $\forall(a \mid a \in S : a + a = a)$ and $\alpha \in \{\emptyset\}$ is commutative $\forall(a, \alpha \mid a \in S, \alpha \in \{\emptyset\} : a\alpha b = b\alpha a)$.

When the set of non-decomposable nonfunctional requirements is not empty (i.e., $N_f \neq \emptyset$) and product families are freely generated from a set of features $\mathbb{F}$, then we define a function μ that provides a quantitative assessment of a product family with regard to a set of nonfunctional requirements. However, prior to that, we remind the reader of the definitions of products and features that were previously given on page 36.

Product [HKM11]: If $a \neq 0$ and if:

- $\forall(b \mid: b \leq a \implies b = 0 \vee b = a)$ and
- $\forall(b, c \mid: a \leq b + c \implies a \leq b \vee a \leq c)$.

Then, a is called a *proper product.*

Feature [HKM11]: Given the divisibility relation | defined as

$$x|y \iff \exists(z, \alpha \mid z \in S \wedge \alpha \in \Gamma : y = x\alpha z)$$

and a proper product a, which satisfies the following:

- $\forall(b \mid: b|a \implies b = 0 \vee b = a)$ and
- $\forall(b, c \mid: a|(b + c) \implies a|b \vee a|c)$

Then, a is called a *feature.*

Definition 1.1 (Quantitative Assessment Function μ)**.** *Let $\big(S, \Gamma, +, 0_s, 1_s\big)$ be the ordered pairs model of an idempotent commutative Γ-semiring where S is freely generated from a set of features $\mathbb{F}$, and Γ is the power set of a set N_f of non-decomposable nonfunctional requirements. For $\alpha \in \Gamma$ and a set of intervals I provided by stakeholders, we define the quantitative assessment function μ_α as follows:*

1. $\mu_\alpha(0_s) = [0, 0]$
2. $\mu_\alpha(1_s) = [N, N]$
3. *For $\alpha = \emptyset$, we have* $\forall(a \mid a \in S - \{0_s\} : \mu_\alpha(a) = [N, N])$
4. *For $\alpha \neq \emptyset$, we have the following cases:*
   (a) $\forall(b \mid b \in \mathbb{F} : \mu_\alpha(b) = b_\alpha)$
   (b) $\mu_\alpha(x + y) = \mu_\alpha(x) \oplus \mu_\alpha(y)$
   (c) $\mu_\alpha(x\alpha y) = \mu_\alpha(x) \odot \mu_\alpha(y)$
   (d) $\mu_\alpha(x\beta y) = [N, N]$ *for* $\beta \neq \alpha$

■

The first two cases are the base cases, the *zero* family 0_s and the *neutral* family 1_s. For every nonfunctional requirement α in Γ, a quantitative weight of $[0, 0]$ is assigned to the assessment of the zero family 0_s, and a full weight of $[N, N]$ is always assigned to the assessment of the neutral family 1_s. For example, the *Purchase-Value-Scope* operationalization in our case study in Figure 4.2 is assumed to hinder the behaviour of the *Business-Management* system and the nonfunctional requirements due to some flaws. Therefore, it is a zero product 0_s and has an assessment of $[0, 0]$ whether we are assessing for performance or security. Whereas the *Advertisement-Management-Interface* operationalization is assumed to perform perfectly in our case study and under any circumstances. Thus, it is considered a neutral product 1_s and has an assessment of $[100, 100]$ for both performance and security nonfunctional requirements.

Apart from the base cases, we have two more special cases. The first one is the quantitative assessment with regard to $\alpha = \{\emptyset\}$. In this case, since we do not have any nonfunctional requirements to consider, all families in S will always have a perfect assessment of $[N, N]$, except for the zero family 0_s since it always has a weight of $[0, 0]$.

For instance, if we are assessing the *Shipping-Gateways* family in Figure 4.1 with no respect to any nonfunctional requirement. We take the following sub-families in S: a = *Fedex*, b = *UPS*, c = *USPS*, d = *Canada-Post*, and e = *Custom-Shipping-Gateways*. Due to the exclusive-or relation between optional features we will get the following:

$\mu(a + b + c + d + e)$

$=<$Definition of $\mu(x + y) >$

$\mu(a) \oplus \mu(b) \oplus \mu(c) \oplus \mu(d) \oplus \mu(ce)$

$=<$Assigning Values$>$

$[100, 100] \oplus [100, 100] \oplus [0, 0] \oplus [100, 100] \oplus [100, 100]$

$=<$Definition of $\oplus >$

$[0, 100]$

As seen previously, all *Shipping-Gateways* options were given neutral assessment values of $[100, 100]$ since no nonfunctional requirement is considered. Except for the the *USPS* company, which was given a zero assessment $[0, 0]$ since it is a zero product 0_s.

The other case is the quantitative assessment with respect to $\alpha \neq \{\emptyset\}$, which is the general case of the proposed approach. In the latter case, a family $a \in S$ can be one of four subsequent cases, a *feature*, an optional composite product family (i.e., $x + y$), a mandatory composite product family with consideration to α operator (i.e., $x\alpha y$), or a mandatory composite product family with consideration to β operator while evaluating for α (i.e., $x\beta y$, $\beta \neq \alpha$).

As for features, a quantitative weight is given by stakeholders and assigned to each feature b in the set of basic features $\mathbb{F}$. Thus, the use of empirical data is crucial in this approach. As indicated in Figures 4.1, 4.2, and 4.3, all bold assessment values are assumed to be assigned by stakeholders. As for the composite families, we first assess the weight of each family independently using the μ function. After that, we calculate the overall weight using either the $\oplus$ operator (for optional compositions) or the $\odot$ operator (for mandatory compositions).

For instance, to calculate the overall performance assessment of the *Fulfillment* family in Figure 4.1, we apply Definition 1.1. Due to the exclusive-or relationship between optional features, we use the binary operation $\oplus$. Assuming we have three sub-families $x, y, z \in S$ where $x =$ *Physical-Goods-Fulfillment*, $y =$ *Electronic-Goods-Fulfillment*, and $z =$ *Services-Fulfillment*. Given $\alpha =$ performance, then:

$\mu_\alpha(x + y + z)$

=<Definition of $\mu(x + y)$ >

$\mu_\alpha(x) \oplus \mu_\alpha(y) \oplus \mu_\alpha(z)$

=<Assigning Values>

$[0, 7] \oplus [4, 16] \oplus [2, 100]$

=<Definition of $\oplus$ >

$[min\ (0, 4), max\ (7, 16)] \oplus [2, 100]$

=<Definition of min and max >

$[0, 16] \oplus [2, 100]$

=< Definition of $\oplus$ >

$[min\ (0,2), max\ (16,100)]$

=< Definition of *min* and *max* >

$[0,100]$

Accordingly, the performance assessment of the *Fulfillment* family includes all the possible performance assessments of the three fulfillment options, starting with the minimum value of 0 and ending with the maximum value of 100.

Likewise, to calculate the overall security assessment of the *Emails* family in Figure 4.1 and due to the inclusive-or relationship between optional features, we use the $\odot$ binary operation. Therefore, assuming we have two sub-families $x, y \in S$ where $x =$ *Personalized* and $y =$ *Response-Tracking*. Given $\alpha =$ security, then:

$\mu_\alpha((1+x)\alpha(1+y))$

=< Definition of $\mu(x\alpha y)$ >

$\mu_\alpha(1+x) \odot \mu_\alpha(1+y)$

=< Definition of $\mu(x+y)$ >

$(\mu_\alpha(1) \oplus \mu_\alpha(x)) \odot (\mu_\alpha(1) \oplus \mu_\alpha(y))$

=<Assigning Values>

$([100,100] \oplus [100,100]) \odot ([100,100] \oplus [1,3])$

=< Definition of $\oplus$ >

$[min\ (100, 100), max\ (100, 100)] \odot [min\ (100, 1), max\ (100, 3)]$

=<Definition of min and max >

$[100, 100] \odot [1, 100]$

=<Definition of $\odot$ >

$[min\ (100, 1), min\ (100, 100)]$

=<Definition of min >

$[1, 100]$

Hence, the overall security assessment of the *Emails* family ranges between 1 and 100. This makes sense as it covers all the possible values in case of choosing only one of the sub-families, both of them, or even none of them.

Another example is to calculate the overall performance of the *Discount-Conditions* family in Figure 4.2. Due to the inclusive-or relationship between both mandatory and optional features, we use the $\odot$ binary operation. Assuming we have three sub-families x = *Product-and-Quantity-Scope*, y = *Time-Scope*, and z = *Purchase-Value-Scope*, and given α = performance, then:

$\mu_\alpha(x\alpha y\alpha(1+z))$

$=<$Definition of $\mu(x\alpha y)>$

$\mu_\alpha(x) \odot \mu_\alpha(y) \odot \mu_\alpha(1+z)$

$=<$Definition of $\mu(x+y)>$

$\mu_\alpha(x) \odot \mu_\alpha(y) \odot (\mu_\alpha(1) \oplus \mu_\alpha(z))$

$=<$Assigning Values$>$

$[44, 90] \odot [13, 70] \odot ([100, 100] \oplus [0, 0])$

$=<$Definition of $\oplus>$

$[44, 90] \odot [13, 70] \odot [min\ (100, 0), max\ (100, 0)]$

$=<$Definition of min and $max>$

$[44, 90] \odot [13, 70] \odot [0, 100]$

$=<$Definition of $\odot>$

$[min\ (44, 13), min\ (90, 70)] \odot [0, 100]$

$=<$Definition of $min>$

$[13, 70] \odot [0, 100]$

$=<$Definition of $\odot>$

$[min\ (13, 0), min\ (70, 100)]$

$=<$Definition of $min>$

$[0, 70]$

The overall performance assessment of the *Discount-Conditions* mostly covers the possible values of the two mandatory sub-families. However, the third sub-family is a zero product whose assessment has no destructive affect since it is optional.

In the language of Γ, we can have terms that involve several operators from Γ (i.e., $a\alpha b\beta c$, where $a, b \in S \wedge \alpha, \beta \in \Gamma \wedge \beta \neq \alpha$). This means that we may encounter more than one nonfunctional requirement in the assessment process (i.e., we take α as security and β as performance). The case of encountering the nonfunctional requirement β while evaluating for the nonfunctional requirement α is illustrated in Definition 1.1. In this case, while evaluating for α, we overlook the non-considered requirement β by assigning a neutral value of $[N, N]$ to the assessment of the product families that use β as a means of connectivity.

1.3 Quantitative Assessment Function $\hat{\mu}_u$

A more complex situation is the quantitative assessment of product families with respect to a set of non-decomposable nonfunctional requirements. We define $\hat{\mu}$ as an expansion of the μ function. It expands from dealing with only one nonfunctional requirement α to a set of nonfunctional requirements $\alpha_1, ..., \alpha_n$.

When given a family of functions $\mu_{\alpha_1}, ..., \mu_{\alpha_n}$ defined according to Definition 1.1, then it is critical to involve stakeholders. Stakeholders have a certain priority or preference associated to each and every nonfunctional requirement. Therefore, whenever a software architect is swamped with many nonfunctional requirements, it is better to prioritize them according to a given preference factor. Accordingly, we have two cases in regards of composite nonfunctional requirements. The first is when no priority is provided and the second is when priority is provided.

In the case where no preference factor is provided, all nonfunctional requirements are treated equally. The assessment is achieved through the use of our *unweighted quantitative assessment function* $\hat{\mu}_u$, which is defined as follows.

Definition 1.2 (Unweighted Quantitative Assessment Function $\hat{\mu}_u$)**.** *Let* $\Gamma = \mathcal{P}(N_f)$*, the power set of a set of non-decomposable nonfunctional requirements to be considered* N_f*. Let* $\big(S, \Gamma, +, 0_s, 1_s\big)$ *be the ordered pairs model of a commutative idempotent* Γ*-semiring where* S *is freely generated from a set of basic features* $\mathbb{F}$*. We define the function:*

$$\hat{\mu}_u : \mathcal{P}(\Gamma) \times S \to I$$

Given $\lambda \in \mathcal{P}(\Gamma)$*, we have:*

1. $\hat{\mu}_u(\lambda, x + y) = \oplus(\alpha | \alpha \in \lambda : \mu_\alpha(x + y))$
2. $\hat{\mu}_u(\lambda, x\lambda y) = \odot(\alpha | \alpha \in \lambda : \mu_\alpha(x\alpha y))$

■

In general, the μ_u function takes a product family in S along with λ, a subset of Γ, to generate a range of possible weight values in the form of an interval I. First, we calculate the overall weight of the considered product family with respect to every α in λ individually using the μ function in Definition 1.1. Then, the $\hat{\mu}_u$ function builds up the results using either the $\odot$ (for mandatory compositions) or $\oplus$ (for optional compositions) operators to reach the overall weight of the product family with respect to all considered nonfunctional requirements.

A noteworthy information is that $\hat{\mu}_u$ is a recursive function as it applies the same calculations recursively for every α in λ.

Let us assess the *Rate-Factors* family in our running case study in Figure 4.1 with regard to both performance and security nonfunctional requirements. Therefore, $\lambda = \{s, p\}$ and we take α as security and β as performance. Due to the exclusive-or relationship between optional features, we use the $\oplus$ operator. Let $a =$ *Quantity-Purchased*, $b =$ *Order-Total*, $c =$ *Weight*, and $d =$ *Product-Classification*. Then:

$\hat{\mu}_u(\lambda, a + b + c + d)$

$=<$Definition of $\hat{\mu}_u >$

$\oplus(\alpha | \alpha \in \lambda : \mu_\alpha(a + b + c + d))$

$=<$Distribution of $\oplus >$

$\mu_\alpha(a + b + c + d) \oplus \mu_\beta(a + b + c + d)$

$=<$Definition of $\mu(x + y) >$

$(\mu_\alpha(a) \oplus \mu_\alpha(b) \oplus \mu_\alpha(c) \oplus \mu_\alpha(d)) \oplus (\mu_\beta(a) \oplus \mu_\beta(b) \oplus \mu_\beta(c) \oplus \mu_\beta(d))$

$=<$Assigning Values$>$

$([6, 29] \oplus [5, 15] \oplus [2, 8] \oplus [7, 14]) \oplus ([3, 5] \oplus [12, 20] \oplus [7, 49] \oplus [2, 4])$

=<Definition of $\oplus$, min, and max >

$[2, 29] \oplus [2, 49]$

=<Definition of $\oplus$ >

$[min\ (2, 2), max\ (29, 49)]$

=<Definition of min and max >

$[2, 49]$

Thereafter, we can say that the overall assessment of the *Rate-Factors* family in our case study in Figure 4.1 with respect to performance and security at the same time is within the range of $[2, 49]$.

Using the previously mentioned example of $N_f = \{s, p\}$ and $\Gamma = \{\{\emptyset\}, \{s\}, \{p\}, \{s, p\}\}$, it is crucial to illustrate the difference between the two looking alike cases of $\alpha = \{s, p\}$ and $\lambda = \{\{s\}, \{p\}\}$. In the earlier case, we are dealing with one nonfunctional requirement $\alpha \in \Gamma$ that is composite. Thus, we have two ways to deal with it. Either we deal with it all as one requirement, create one table of stakeholders' input data for it, and use the quantitative assessment function μ to assess its weight value. Or, we can deal with it separately as two different requirements and address both of them using the quantitative assessment function $\hat{\mu}$. However, in the latter case, we have $\lambda \in \mathcal{P}(\Gamma)$. Thus, we definitely deal with them as two distinct nonfunctional requirements, which use the quantitative function $\hat{\mu}$ to assess their weights.

1.4 Weighted Quantitative Assessment Function μ_w

The second case of assessing product families with respect to a set of non-decomposable nonfunctional requirements is when given a preference factor P for all considered nonfunctional requirements. The preference factor P is represented by assigning weight values to the preference or priority of the nonfunctional requirements.

Given $\mathcal{P}(N_f)$, the power set of the set of non-decomposable nonfunctional requirements to be considered, we define the preference factor P as follows:

$$P : \Gamma \rightarrow [0...1]$$

Such that: $\sum_{\alpha \in \Gamma} P(\alpha) = 1$

A mapping is carried out between every considered nonfunctional requirement $\alpha \in \Gamma$ and the weight value of its assigned preference [0...1]. With 0 being a non-critical requirement whereas 1 indicates the maximum critical need of that requirement. This assignment is given by stakeholders.

In our *Business-Management* case study illustrated in Figures 4.1, 4.2, 4.3, we assign weight values of 0.4 and 0.6 to the preference of the considered nonfunctional requirements, security and performance (respectively). Notice that the total weight value is $P = 0.4 + 0.6 = 1$.

Given P, the weight value of the preference for the considered nonfunctional requirements, we define:

$$\theta : P \times I \rightarrow I$$

Where $\sum_{\alpha \in \Gamma} P(\alpha) = 1$, and such that:

1. $\theta(0, [a, b]) = [N, N]$
2. $\theta(n, [a, b]) = [na, nb]$

A special case is when some nonfunctional requirement has 0 as preference weight. For that special case, the above function θ is defined. In general, assuming we have a nonfunctional requirement α, the function θ takes the assessment weight μ_α of that requirement and multiplies it with its associated preference weight P_α. However, when having a zero priority (i.e., $P_\alpha = 0$), the function θ returns a value of $[N, N]$ to avoid multiplying the endpoints of the given interval by 0, which will result into an interval of zeros. Otherwise, the function θ returns the interval obtained by multiplying the endpoints of the given interval by the given preference weight. The point of the function θ is to avoid the complete elimination of requirements that have no preference.

Finally, the *weighted quantitative assessment function* $\hat{\mu}_w$ is defined as follows.

Definition 1.3 (Weighted Quantitative Assessment Function $\hat{\mu}_w$)**.** *Let $\Gamma = \mathcal{P}(N_f)$, the power set of a set N_f of non-decomposable nonfunctional requirements to be considered. Let $\big(S, \Gamma, +, 0_s, 1_s\big)$ be the ordered pairs model of a commutative idempotent Γ-semiring where S is freely generated from a set of basic features $\mathbb{F}$. We define the function:*

$$\hat{\mu}_w : \mathcal{P}(\Gamma) \times S \to I$$

Given $\lambda \in \mathcal{P}(\Gamma)$, we have:

1. $\hat{\mu}_w(\lambda, x + y) = \oplus(\alpha | \alpha \in \lambda : \theta(P(\alpha), \mu_\alpha(x + y)))$

2. $\hat{\mu}_w(\lambda, x\lambda y) = \odot(\alpha | \alpha \in \lambda : \theta(P(\alpha), \mu_\alpha(x\alpha y)))$

■

The function $\hat{\mu}_w$ in Definition 1.3 does the exact same recursive calculations as $\hat{\mu}_u$ in Definition 1.2. The only added difference is that $\hat{\mu}_w$ takes the weight value of the preference P into consideration. In general, $\hat{\mu_w}$ takes a product family in S along with λ, a subset of Γ, to generate an interval I, after taking into account the weight value of the preference P.

First, we calculate the overall assessment of the considered product family with respect to every α in λ individually using the μ function in Definition 1.1. Then, the function θ multiplies the resulted assessment of every α by the weight value of its preference P_α. Finally, the $\hat{\mu}_w$ function adds up the results using either the $\odot$ (for mandatory compositions) or $\oplus$ (for optional compositions) operators to reach the overall assessment of the product family with respect to all considered prioritized nonfunctional requirements.

In our case study in Figure 4.3, we take the following families in S: a = *Customer-Preference*, b = *Personal-Information*, c = *Demographics*, d = *Previous-Purchases*, e = *Shopping-Cart-Content*, f = *Wish-List-Content*, g = *Previously-Visited-Pages*, h = *Date-and-Time*, and i = *Custom-Targeting-Criteria*. Also, let $\lambda = \{s, p\}$ and we take $\alpha = \{s\}$ and $\beta = \{p\}$. Due to the exclusive-or relationship between optional features, we use the $\oplus$ operator. To calculate the overall assessment of *Targeting-Criteria* with regard to both performance and security at the same time and while considering the weight value of the preference P, we do the following:

$\hat{\mu}_w(\lambda, a+b+c+d+e+f+g+h+i)$

=<Definition of $\hat{\mu}_w$ >

$\oplus(\alpha | \alpha \in \lambda : \theta(P(\alpha), \mu_\alpha(a+b+c+d+e+f+g+h+i)))$

=<Distribution of $\oplus$ >

$\theta(P(\alpha), \mu_\alpha(a+b+c+d+e+f+g+h+i)) \oplus \theta(P(\beta), \mu_\beta(a+b+c+d+e+f+g+h+i))$

=<Definition of $\mu(x+y)$ >

$\theta(P(\alpha), \mu_\alpha(a) \oplus \mu_\alpha(b) \oplus \mu_\alpha(c) \oplus \mu_\alpha(d) \oplus \mu_\alpha(e) \oplus \mu_\alpha(f) \oplus \mu_\alpha(g) \oplus \mu_\alpha(h) \oplus \mu_\alpha(i)) \oplus$
$\theta(P(\beta), \mu_\beta(a) \oplus \mu_\beta(b) \oplus \mu_\beta(c) \oplus \mu_\beta(d) \oplus \mu_\beta(e) \oplus \mu_\beta(f) \oplus \mu_\beta(g) \oplus \mu_\beta(h) \oplus \mu_\beta(i))$

=<Assigning Values>

$\theta(0.4, [53, 92] \oplus [71, 73] \oplus [30, 89] \oplus [9, 81] \oplus [1, 30] \oplus [4, 60] \oplus [3, 10] \oplus [2, 2] \oplus [10, 100]) \oplus$
$\theta(0.6, [6, 15] \oplus [7, 16] \oplus [8, 17] \oplus [1, 10] \oplus [2, 11] \oplus [4, 13] \oplus [9, 18] \oplus [3, 12] \oplus [5, 14])$

=< Definition of $\oplus$, min, and max >

$\theta(0.4, [1, 100]) \oplus \theta(0.6, [1, 18])$

=< Definition of θ >

$[0.4, 40] \oplus [0.6, 10.8]$

=< Definition of $\oplus$ >

$[min\ (0.4, 0.6), max\ (40, 10.8)]$

=< Definition of min and max >

$[0.4, 40]$

Thereafter, we can say that the overall assessment of *Targeting-Criteria* with respect to performance and security requirements at the same time along with their preference weight values is within the range of $[0.4, 40]$.

One last example is the assessment the product family *Targeting* in Figure 4.1 of our case study with regard to both performance and security. We take $\alpha = \{s\}$ and $\beta = \{p\}$ and therefore $\lambda = \{s, p\}$. Due to the inclusive-or relationship we use the $\odot$ operator. We also take the following sub-families in S: a = *Targeting-Mechanism*, b = *Targeting-Criteria*, c = *Campaigns*, and d = *Display-and-Notification*. Given the preference weight values $P(s) = 0.4$ and $P(p) = 0.6$, we have:

$\hat{\mu}_w(\lambda, a\alpha b\alpha(1+c)\alpha d)$

=<Definition of $\hat{\mu}_w$ >

$\odot(\alpha|\alpha \in \lambda : \theta(P(\alpha), \mu_\alpha(a\alpha b\alpha(1+c)\alpha d)))$

=<Distribution of $\odot$ >

$\theta(P(\alpha), \mu_\alpha(a\alpha b\alpha(1+c)\alpha d)) \odot \theta(P(\beta), \mu_\beta(a\beta b\beta(1+c)\beta d))$

=<Definition of $\mu(x\alpha y)$ >

$\theta(P(\alpha), \mu_\alpha(a) \odot \mu_\alpha(b) \odot \mu_\alpha(1+c) \odot \mu_\alpha(d)) \odot \theta(P(\beta), \mu_\beta(a) \odot \mu_\beta(b) \odot \mu_\beta(1+c) \odot \mu_\beta(d))$

=<Definition of $\mu(x+y)$ >

$\theta(P(\alpha), \mu_\alpha(a) \odot \mu_\alpha(b) \odot (\mu_\alpha(1) \oplus \mu_\alpha(c)) \odot \mu_\alpha(d)) \odot \theta(P(\beta), \mu_\beta(a) \odot \mu_\beta(b) \odot (\mu_\beta(1) \oplus \mu_\beta(c)) \odot \mu_\beta(d))$

=<Assigning Values>

$\theta(0.4, [0,30] \odot [1,100] \odot ([100,100] \oplus [50,100]) \odot [1,100]) \odot \theta(0.6, [0,80] \odot [1,18] \odot ([100,100] \oplus [16,20]) \odot [1,100])$

=<Definition of $\oplus$, min, and max >

$\theta(0.4, [0,30] \odot [1,100] \odot [50,100] \odot [1,100]) \odot \theta(0.6, [0,80] \odot [1,18] \odot [16,100] \odot [1,100])$

=<Definition of $\odot$ and min >

$\theta(0.4, [0,30]) \odot \theta(0.6, [0,18])$

=<Definition of θ >

$[0,12] \odot [0,10.8]$

=<Definition of $\odot$ and min >

$[0,10.8]$

The overall assessment of the *Targeting* family with respect to performance and security at the same time along with their preference weight value falls within the range of $[0, 10.8]$.

1.5 Conclusion

Measurement is essential to many systems in our lives. As put in [FP97], "You cannot control what you cannot measure". Thus, we should be creating ways to measure our world and therefore enhance it. However, the measurement process is not defined in a clear-cut sense and this raises a concern.

An entity is an object such as a person, a place, or an event. Whereas an attribute is a property of an entity. For example, if the entity is a journey then an attribute could be the cost of that journey. Accordingly, a measurement process is defined as "the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules" [FP97]. For example, to describe the cost of a journey we will assign a number of dollars as a measurement. So many loose measurement terminology are still commonly accepted and it is our rule as scientists to overcome this issue and improve measurement techniques.

Software engineering aggregates a number of engineering techniques to implement and maintain software products in scientific and controlled way. In software industry, we want to be able to measure the quality of a software products, and this is why the measurement process was introduced to software engineering. Software measurement in software engineering is a collection of topics called *software metrics*. Thus, a software metric is "a term that embraces many activities, all of which involve some degree of software measurement" [FP97]. Examples of these topics include: cost and effort estimation, data collection, quality models and measures, and much more. Each of these measurements has a measurement model.

Our attempt to quantify nonfunctional requirements matches the measurement definition in terms of assigning values to attributes. It is a software measurement that falls under the *quality models and measures*. Theses models (just like our model) consists of high-level quality factors that need to be quantified. These quality factors are decomposed into lower-level criteria that are easier to understand, measure, and deal with. The resulted tree-like structure contains correlations between factors and their descendants, and thus a factor is measured through the measurement of these relationships.

A model [FP97] is an abstraction of reality that takes the form of equations, mappings, or even diagrams. Models enable us to view a particular aspect of this reality from a certain point of view, and they also allow us to examine the relationships between different component parts. For example, our proposed model allows us to view the relationships between components in terms of considered nonfunctional requirements. Unfortunately, when modelling we often focus more on the mathematical system and manipulating numbers while disregarding the empirical system and relationship among entities. A part of modelling reality is to model its attributes and entities as mentioned in the above measurement definition. Once we have the model along with its entities and attributes, we start the measuring.

We define the measures based on the attributes, and they can be either direct or indirect [FP97]. *Direct measurement* directly maps an entity to a numerical value without the use of any other entity or attribute. On the other hand and in the case of complex relationships, an *indirect measurement* is established by combining several sub-attributes and sub-aspects, and a model is needed to show how the combination is established. Hence, we can say that indirect measurements show interactions between direct measurements. Our proposed model is an example of an indirect measurement. It measures the assessment of nonfunctional requirements in product families through the assessments of sub-families and sometimes the level of preference of the requirements.

Most measurements are for existing entities. However, we sometimes would like to predict an entity and hence measure it. Such thing is called *prediction measurement*. As stated in [FP97], "A prediction system consists of a mathematical model together with a set of prediction procedures for determining unknown parameters and interpreting results (Littlewood, 1988)". Therefore, for prediction measurement we can create a mathematical model of the factors that affect the entity to show the relations between the attributes to be predicted and the affecting attributes. After that, we understand the model and use it to predict the measurement. Similarly, we predict that nonfunctional requirements can be measured and hence created the proposed mathematical model following the exact same step.

As mentioned earlier, in direct measurements we assign mappings or representations from an empirical relation system (domain) to a numerical relation system (range). Therefore, we can manipulate data in the numerical system to measure the attributes in the empirical one. The mapping along with both the empirical and numerical relation systems are called a *measurement scale* [FP97]. There are five major types of scales: Nominal, Ordinal, Interval, Ratio, and Absolute. The ordinal scale maintains ordering and thus carries more information about the entities than the nominal scale. However, the interval scale carries even more information, which makes it more powerful. It preserves order, preserves differences, captures interval sizes between classes, and accepts addition an subtraction. During the assignment, we have to assure that the suitable kind of measurement scale is assigned. For our model, we used the interval scale.

To summarize, our model is an indirect measurement to measure a predictable entity (nonfunctional requirements) using the interval measurement scale.

In order to measure nonfunctional requirements satisfactorily, we created functions like μ and $\hat{\mu}$, operators like $\oplus$ and $\odot$, and relations like $\leq_I$. However, we were dedicated during the creation to define them as sound measures. Some software engineers claim that quality attributes (nonfunctional requirements) such as performance and reliability cannot be measured. This claim is valid so far due to the lack of measurement techniques. However, once a model proposal (like our model) is made, so many discussions will carry out the work to a better model.

Chapter 5

Future Work

This chapter highlights briefly the possible future work directions in Section 5.1. In Section 5.2, we discuss a potential proper model for the proposed Γ-semiring structure. Finally, we present some works suggested for further investigations in Section 5.3.

1 Directions For Future Work

Our future work will evolve in the following main directions. First, enhancing the practical assessment of our approach is intended to be a main subject of future work. We will try to apply the proposed algebraic technique to a concrete model, such as the model illustrated briefly in Section 5.2.

Current work concerns nonfunctional requirements that can be expressed in a quantitative manner, such as security and performance. However, whereas some nonfunctional requirements are measurable (by observation) like the level of performance, other requirements have precise measurement metrics, such as the cost and energy consumption. Therefore, another future work direction is extending the approach to cover other types of quantitative nonfunctional requirements.

Finally, this research considers only frameworks that model relationships between nonfunctional requirements and their decomposed off-spring operationalizations. However, literature is enriched with other models. There are models [KS00] that represent relationships between nonfunctional requirements and design decisions. Whereas other models [Ngu09] [CdPL04] view functional and nonfunctional requirements as two distinct models to apply the separation of concerns concept, and to trace the impact caused by functional requirements over the nonfunctional requirements and hence the overall quality of the system. These models can form another direction for future work. Existing approaches which support functional verification can be integrated with our approach to nonfunctional assessment to provide environments in which quality assessments can be performed for software product families.

2 Notes on a Future Model for Γ-Semirings

Modelling the mathematical structure of Γ-semirings has been scarce in the literature. In this section, we present a potential implementation or a model for the proposed algebra (Γ-semiring structure) that can be further expanded and used for assessing nonfunctional requirements. The model is inspired by an example in [SD04]. The example presented a Γ-semiring S with matrices as its elements. A proposition $a\alpha b$ denotes the matrix product of two matrices, S and Γ, where $a, b \in S$ and $\alpha \in \Gamma$. S and Γ are two additive commutative semigroups that are presented as two matrices of size 2×3 and 3×2 (respectively) over the set of non-negative rational numbers Q_0^+.

$$\text{Assuming } a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix}, b = \begin{bmatrix} 0 & 0 & \frac{1}{3} \\ 0 & 1 & 0 \end{bmatrix}, \text{ and } \alpha = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

$$\text{Then, } a\alpha b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & \frac{1}{3} \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 \end{bmatrix}.$$

The use of matrices is a simple direct potential implementation of Γ-semirings. As seen in the example, products $a, b \in S$ are represented through matrices, and thus all values within those particular matrices can be thought of as evaluations of features within products a and b. Initially, we can say that in each product matrix, rows resemble components of the system where columns resemble the considered nonfunctional requirements. Therefore, each entry in the matrix represent the evaluation of a specific nonfunctional requirement in a specific system component. Moreover, all considered NFR-operators (i.e., $\alpha \in \Gamma$) are presented as matrices as well. The matrix entries here represents different ways of interactions between the system components. Finally, the inner calculations between these matrices to carry out the result of the proposition $a\alpha b$ can be viewed as quantitative assessments of the different ways of feature-interactions among products a and b via the NFR-operator α. For a visual representation of feature-interactions, we have the following example:

Assuming $a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$, $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, and $\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 \end{bmatrix}$.

Then, $a\alpha b = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \times \begin{bmatrix} \alpha_1 & \alpha_2 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1\alpha_1 b_1 + a_1\alpha_2 b_2 \\ a_2\alpha_1 b_1 + a_2\alpha_2 b_2 \end{bmatrix}$.

3 More Suggested Directions for Future Work

The following are works that present relationships between nonfunctional requirements and other elements such as functional requirements or design decisions. These works are referenced as they could be useful for future work.

3.1 A Goal-Based Model

A goal-based model is proposed in [Ngu09]. In this model, two separate AND/OR trees are constructed, one for functional requirements (hard-goals) and another for nonfunctional requirements (soft-goals). These two trees are connected throughout the same kind of correlations that link between nonfunctional requirements and their refinements: Break $--$, Hurt $-$, Unknown ?, Help $+$, and Make $++$. This representation is used to examine and trace the impact of functional changes over nonfunctional soft-goals. As suggested in [Ngu09], this combination of hard-goals, soft-goals, and correlations can form a matrix in which hard-goals are listed as rows, soft-goals as columns, and correlations as cell entries. These correlations are of qualitative nature and have qualitative values. However, they can have quantitative values drawn from the levels of priority of soft-goals assigned by the stakeholders. Based on the values in the matrix, a software engineer can apply trade-off analysis and determine the possibility of constructing single products within a software product family.

3.2 The Language Extended Lexicon

The software development process is viewed in [CdPL04] as composed of two distinct evolutionary perspectives: one focuses on functional aspects and the other focuses on nonfunctional aspects. Having an approach that deals with these two different processes separately is good for many reasons. One of which is the fact that requirements changes can be triggered by either functional or nonfunctional aspects, thus the separation of concerns eases the evolution aspect. Another reason is for the process to be used in legacy system. It also allows us to detect any design inconsistency among interdependencies when analyzing NFR-graphs.

In the Language Extended Lexicon (LEL) strategy, we first need a lexicon to represent the common vocabulary of the requirements domain. After we build it, functional and nonfunctional models are built separately yet concurrently. When nonfunctional requirements are added to the lexicon, some NFR-implementation solutions are also added. Thereafter, NFR-graphs are then constructed and extended to the strategy. These graphs are driven from the Chung's NFR-framework [CdPL09] which proposes the goal-oriented model. It enables a deeper level of reasoning about nonfunctional requirements in order to capture conflicts and apply trade-offs. Finally, we integrate the two models and address different feedbacks during the evolution of the process.

3.3 Definition Hierarchy

Software reuse is a key to high quality and productivity. Product families are among the best software reuse approaches. However, they suffer the lack of support by many current practices in requirements engineering [KS00]. Definition hierarchy provides a requirements engineering solution to the problem [Ngu09]. It models nonfunctional requirements for product families and helps to reflect the variation of products within one family by organizing the requirements of all different products into the same definition hierarchy.

As stated earlier, most requirement analysis approaches divide requirements into functional and nonfunctional. Nevertheless, definition hierarchy divide requirements a bit differently into *design objectives* and *design decisions* [KS00]. Design objectives act as the nonfunctional requirements as they are basically the essence of the user requirements. They state how the functionality of the system should be. These objectives can be either general and very abstract requirements that are applicable to the whole system, or they can be design specific just as required by the customer. Design decisions are a bunch of features that anticipate, during the early requirement analysis phase, how the resulted product is going to be. Ideally, no design decisions should be made during requirements analysis, however in practice, many decisions are actually taken during that phase. With product families, it is relatively easy to tell whether or not a product belongs to a certain family by assuring it conforms to the family architecture. This facilitates the reflection of the implementation to the requirement analysis phase, and therefore the creation of design decisions in product families. After gathering the requirements we divide them by their type into functional and nonfunctional, or by their domain such as security, reliability and so on. However, in both cases requirements still vary in term of the level of detail. Here emerges the need to using the definition hierarchy.

Kuusela and Savolainen illustrated in [KS00] that the definition hierarchy is a logical AND tree which is composed of two major elements: *nodes* and *edges*. Nodes represent both design objectives (NFRs) and design decisions (features) where design objectives are defined by other design objectives and design decisions. Thus, the child node can be either a decision satisfying the parent node or just a sub-objective that helps defining its parent requirement. There is also the root node that resembles the main concept of the tree and the purpose of the system. On the other hand, design objectives and decisions depend on one another. Their dependency is design specific and it is expressed throughout edges. When an edge connect two objectives together then it is a refinement between them, but when it links an objective to a decision then it means that the objective is partially satisfied by that decision.

Priorities of the requirement are assigned to each node in a product family. If a feature or a decision has no impact whatsoever on an NFR or an objective, its priority is equal to zero. The priority of a child cannot exceed that of its parent [Ngu09]. Like many other NFR graphs, definition hierarchy helps to detect conflicts and inconsistencies among requirements structure.

Bayesian Belief Network (BBN) and Extended PLUS [Ngu09] are also used for modelling nonfunctional requirements in product families. Thus, they relate NFRs to software design and are useful for future work.

Appendix A

1 Chapter 3 Proofs

To prove that $(I, \leq_I)$ is a preorder, we have to prove that $\leq_I$ is reflexive and transitive. Let i_n be $[a_n, b_n]$, where $a_n, b_n \in \mathbb{R}^+$, then:

1. $\forall(i_1 \;\mid\; i_1 \in I \;:\; i_1 \leq_I i_1)$

   $\Longleftarrow$ <Definition of $\leq_I$>

   $\forall(i_1 \;\mid\; i_1 \in I \wedge i_1 = [a_1, b_1] \;:\; b_1 \leq b_1)$

   $\Longleftarrow$ <Reflexivity of $\leq$>

   True

2. $\forall(i_1, i_2, i_3 \mid i_1, i_2, i_3 \in I : i_1 \leq_I i_2 \wedge i_2 \leq_I i_3 \implies i_1 \leq_I i_3)$

$\Longleftarrow$ <Definition of $\leq_I$>

$\forall(i_1, i_2, i_3 \mid i_1, i_2, i_3 \in I \wedge i_1 = [a_1, b_1] \wedge i_2 = [a_2, b_2] \wedge i_3 = [a_3, b_3] : b_1 \leq b_2 \wedge b_2 \leq b_3 \implies b_1 \leq b_3)$

$\Longleftarrow$ <Transitivity of $\leq$>

True

From 1 and 2, $(I, \leq_I)$ is called a pre-ordered set.

To prove that I is closed with respect to $\oplus$, we need to prove that $\forall(i_1, i_2 \mid i_1, i_2 \in I : i_1 \oplus i_2 \in I)$. Let $[a_1, b_1]$ and $[a_2, b_2] \in I$, then:

$[a_1, b_1] \oplus [a_2, b_2] \in I$

$\Longleftrightarrow$ <Definition of $\oplus$ >

$[min\ (a_1, a_2), max\ (b_1, b_2)] \in I$

$\Longleftrightarrow$ <Definition of I >

$0 \leq min\ (a_1, a_2) \leq max\ (b_1, b_2) \leq N \wedge min\ (a_1, a_2), max\ (b_1, b_2) \in \mathbb{R}^+$

$\Longleftrightarrow$ <Transitivity of $\leq$ and the fact that $0 \leq a_1, b_1, a_2, b_2 \leq N$ >

True

To prove that I is closed with respect to $\odot$, we need to prove that $\forall(i_1, i_2 \mid i_1, i_2 \in I \,:\, i_1 \odot i_2 \in I\,)$. Let $[a_1, b_1]$ and $[a_2, b_2] \in I$, then:

$[a_1, b_1] \odot [a_2, b_2] \in I$

$\Longleftrightarrow$ <Definition of $\odot$ >

$[min\ (a_1, a_2), min\ (b_1, b_2)] \in I$

$\Longleftrightarrow$ <Definition of I >

$0 \leq min\ (a_1, a_2) \leq min\ (b_1, b_2) \leq N \wedge min\ (a_1, a_2), min\ (b_1, b_2) \in \mathbb{R}^+$

$\Longleftrightarrow$ <Transitivity of $\leq$ and the fact that $0 \leq a_1, b_1, a_2, b_2 \leq N$ >

True

Bibliography

[AK12] Amy Affleck and Aneesh Krishna. Supporting quantitative reasoning of non-functional requirements: A process-oriented approach. In *2012 International Conference on Software and System Process, ICSSP 2012 - Proceedings*, pages 88–92, General Post Office, P.O. Box 30777, NY 10087-0777, United States, June 2012. Association for Computing Machinery.

[BKJ09] Christopher Burgess, Aneesh Krishna, and Li Jiang. Towards optimising non-functional requirements. In *QSIC 2009 - Proceedings of the 9th International Conference on Quality Software*, number 5381441, pages 269–277, Jeju, Korea, August 2009. IEEE, IEEE Computer Society, Piscataway, NJ, United States.

[BLP05] Stan Bühne, Kim Lauenroth, and Klaus Pohl. Modelling requirements variability across product lines. In *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 41–50. IEEE, Institute of Electrical and Electronics Engineers Computer Society, 2005.

[CBLA08] Luiz Marcio Cysneiros, Karin K. Breitman, Claudia Lopez, and Hernan Astudillo. Querying software interdependence graphs. In *32nd Annual IEEE Software Engineering Workshop, SEW-32 2008*, number 5328411, pages 108–112, Kassandra, Greece, October 2008. IEEE, IEEE Computer Society, Piscataway, NJ, United States.

[CdPL04] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Nonfunctional requirements: From elicitation to conceptual models. *IEEE Transactions on Software Engineering*, 30:328–350, May 2004.

[CdPL09] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On nonfunctional requirements in software engineering. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5600 LNCS, pages 363–379. Springer Verlag, Heidelberg, Germany, 2009.

[CHSB$^+$05] Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhanskaya, and Selvia Christina. Goal-centric traceability for managing non-functional requirements. In *Proceedings - 27th International Conference on Software Engineering, ICSE 2005*, volume 2005 of *27*, pages 362–371, Saint Louis, MO, United states, May 2005. International Conference on Software Engineering ICSE, Institute of Electrical and Electronics Engineers Computer Society.

[CNYM00] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Springer US, 2000.

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, April 1993.

[ES08] Sima Emadi and Fereidoon Shams. An approach to non-functional requirements analysis at software architecture level. In *Proceedings - 2008 IEEE 8th International Conference on Computer and Information Technology, CIT 2008*, pages 736–741, Sydney, NSW, Australia, July 2008. IEEE, Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ, United States.

[FP97] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company, Boston, second edition edition, 1997.

[GF94] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering (Cat. No.94TH0613-0)*, 1, pages 94–101, Colorado Springs, CO, USA, April 1994. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA.

[GS11] Carlo Ghezzi and Amir Molzam Sharifloo. Quantitative verification of non-functional requirements with uncertainty. *Advances in Intelligent and Software Computing*, 97:47–62, 2011.

[GS13] Carlo Ghezzi and Amir Molzam Sharifloo. Model-based verification of quantitative non-functional properties for software product lines. *Information and Software Technology*, 55:508–24, March 2013.

[GW01] David J. Grimshaw and Godfrey W.Draper. Non-functional requirements analysis: Deficiencies in structured methods. *Information and Software Technology*, 43:629–634, October 2001.

[HKM11] Peter Höfner, Ridha Khedri, and Bernhard Möller. An algebra of product families. *Software and Systems Modeling*, 10:161–182, May 2011.

[JFS06] Ivan J. Jureta, Stéphane Faulkner, and Pierre-Yves Schobbens. A more expressive softgoal conceptualization for quality requirements analysis. In *Conceptual Modeling - ER 2006 - 25th International Conference on Conceptual Modeling, Proceedings*, volume 4215 LNCS, pages 281–295. Springer Verlag, November 2006.

[KS00] Juha Kuusela and Juha Savolainen. Requirements engineering for product families. In *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, pages 61–9, Limerick, Ireland, June 2000. IEEE, ACM, New York, NY, USA.

[Kur21] C. Kuratowski. Sur la notion d'ordre dans la theorie des ensembles. pages 161–171, 1921.

[Lau06] Sean Quan Lau. Domain analysis of e-commerce systems using feature-based model templates. Master's thesis, Waterloo University, Waterloo, Ontario, Canada, 2006.

[LK98] Jonathan Lee and Jong-Yih Kuo. New approach to requirements trade-off analysis for complex systems. *IEEE Transactions on Knowledge and Data Engineering*, 10:551–62, July-August 1998.

[MA09] Sonia Montagud and Silvia Abrahão. Gathering current knowledge about quality evaluation in software product lines. In *Proceedings of the 13th International Software Product Line Conference*, pages 91–100, 2009.

[MCN92] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18:483–497, June 1992.

[Ngu09] Quyen L. Nguyen. Non-functional requirements analysis modeling for software product lines. In *Proceedings of the 2009 31st International Conference on Software Engineering and ICSE Workshops - 2009 ICSE Workshop on Modeling in Software Engineering, MiSE 2009*, number 5069898, pages 56–61, Vancouver, BC, Canada, May 2009. IEEE, IEEE Computer Society, Piscataway, NJ, United States.

[PB88] Colin Potts and Glenn Bruns. Recording the reasons for design decisions. In *Proceedings - 10th International Conference on Software Engineering.*, pages 418–427, Singapore, 1988. IEEE, IEEE, New York, NY, USA.

[PLZ09] Xin Peng, Seok-Won Lee, and Wen-Yun Zhao. Feature-oriented non-functional requirement analysis for software product line. *Journal of Computer Science and Technology*, 24:319–338, March 2009.

[PS90] Adam A. Porter and Richard W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7:46–54, March 1990.

[RG11] A. Ananda Rao and M. Gopichand. Four layered approach to non-functional requirements analysis. *IJCSI International Journal of Computer Science Issues*, 8:371–379, November 2011.

[Rya00] Andrew J. Ryan. An approach to quantitative non-functional requirements in software development. *CiteSeer*, December 2000.

[SCCotICS90] USA Standards Coordinating Committee of the IEEE Computer Society. IEEE standard glossary of software engineering terminology, December 1990.

[SD04] S. K. Sardar and U. Dasgupta. On primitive gamma-semirings. *Journal of Mathematics, Novi Sad*, 34(1):1–12, 2004.

[SM98] Alistair G. Sutcliffe and Shailey Minocha. Scenario-based analysis of non-functional requirements. Technical report, The European Commission ESPRIT 21903 'CREWS' (Cooperative Requirements Engineering With Scenarios), Centre for HCI Design, School of Informatics, City University, Northampton Square, London, UK, 1998.

[SOMZ05] Juha Savolainen, Ian Oliver, Mike Mannion, and Hailang Zuo. Transitioning from product line requirements to product line architecture. In *Proceedings of the 29th Annual International Computer Software* *and Applications Conference*, pages 186–195. IEEE Computer Soiety, July 2005.

[TFQ06] Lixin Tao, Xiang Fu, and Kai Qian. *Software Architecture Design - Methodology and Styles*. Stipes Publishing L.L.C., 2006.

[Tha02] Richard H. Thayer. Software system engineering: A tutorial. 35:68–73, April 2002.

[UK11] Mahrukh Umar and Naeem Ahmed Khan. Analyzing non-functional requirements (nfrs) for software development. In *ICSESS 2011 - Proceedings: 2011 IEEE 2nd International Conference on Software Engineering and Service Science*, number 5982328, pages 675–678, Beijing, China, July 2011. IEEE, IEEE Computer Society, Piscataway, NJ, United States.

[vL01] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the IEEE 5th International Conference on Requirements Engineering*, pages 249–261, Toronto, ON, Canada, August 2001. IEEE, Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ, United States.

[Yu97] Eric S. K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering (Cat. No.97TB100086)*, pages 226–35, Annapolis, MD, USA, January 1997. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA Press.

[Zav97] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys*, 29:315–321, December 1997.