Semantic Component Selection Based on Non-Functional Requirements

Amir Kanan Kashefi

The School of Computer Science
The University of Adelaide

January 31, 2018

Contents

Figures

Tables

Abstract

Reusing existing software components in place of requiring the implementation of new components can reduce the complexity of the software development process. However, for a software component to be effectively identified and selected for reuse, we need a good understanding of both the functional and non-functional requirements of the component needed, and the components available. Functional requirements specify what a software component does and non-functional requirements specify how a software component achieves its goals.

Non-functional requirements are typically complex, and difficult to both understand and effectively articulate. Requirements engineering provides a solution to easing this process, and involves performing the following reasoning steps: elicitation, analysis and description. However, the output of these steps is based on reasoning that requires manual, expensive and error-prone techniques. To solve such drawbacks, this thesis describes a framework that provides the necessary tools and techniques for automating reasoning including: an ontology for non-functional requirements as a conceptual model for reasoning; and a search algorithm that matches the best component according to the reasoning process outputs. To validate our framework, we develop an implementation that supports semantic search within a repository to locate matches based on a user query, validated with experimental findings on a repository consisting of 50 individual component descriptions. Our findings demonstrate the benefit obtained from using an ontology, by minimizing the cost and complexity of analysing non-functional requirements. Our algorithm is capable of running a complex query, for example, supporting 5 non-functional requirements with total 16 prerequisites against a repository of 1000 components can run in 1750 second. It would be impossible for a field expert to compute a complex query in this time frame.

Declaration

Signed:

Amir Kanan Kashefi

January, 2018

Acknowledgment

I would like to thank my wife who has supported me with a great deal of love. This work would not have been completed without her.

I would like to thank my family for their encouragement and support during my study. They gave me hope during hard research times.

Finally, I would like to thank my supervisors, Professor Katrina Falkner, Associate Professor Nickolas Falkner and Dr Claudia Szabo for their ongoing support, supervision and advice.

1 Introduction

As technology has become increasingly pervasive in our society, we have seen a corresponding increase in software complexity, size and the need for agility in software evolution. Accordingly, software development companies are seeking techniques to increase their productivity, to cope with this increased complexity and to adapt quickly to changing requirements and markets [1, 2].

It is neither feasible nor economical to rebuild new systems every time significant changes are made to software system requirements. Moreover, adapting a large monolithic system, where understanding of full system operation and requirements is difficult, to new environments and situations is hard if not impossible [3, 4]. For example, in 1994 a US air traffic control system was overdue for replacement by a new, modern system, due to reliability concerns. The federal aviation administration worked on this system evolution for more than 10 years without success, producing a new system with a monolithic structure of more than a million lines of code, and riddled with bugs. The structure of the new system made it impossible to put it into operation, hard to maintain and further evolve [3, 4] **.**

When software developers focus their attention on the elements of writing code, testing their code and fixing errors, rather than systematic overall design, there is a considerable risk involved in code maintenance and evolution. A small change within one part of the software system may require other parts to change due to the interdependencies within the system, potentially escalating a small change to significant system restructure. This process is time-consuming, difficult to understand and increases the overall cost of the project.

Therefore, we should aim to build software systems that are evolvable and geared for change. Software systems must be composed of distinct software components that can be easily reused and replaced. By adopting such techniques, we are better placed to keep pace with ever changing and increasing requirement on software systems [3].

It is imperative that all software producers introduce specific design and development discipline and strategies to ensure that their software development process is efficient, and can cope with evolutionary needs. This discipline should facilitate the development of modules as reusable entities, where the maintenance and upgrading of the software system as a whole becomes more efficient [5] through the individual customisation, testing, debugging or replacement of its modules.

In this chapter, we will motivate the need for new approaches to supporting modular software development, extending current methodologies. This chapter will cover key definitions and requirements of the proposed approach, and this thesis will explore a case study of how this approach can support benefit in software development in contrast to traditional development approaches [6-9].

A commonly used approach for software development is the *Greenfield* approach [10]. The Greenfield approach consists of developing all required software elements without drawing on previously created software, excluding the possibility of reusing any existing software systems. It typically follows a set of specific activities, or phases, such as *analysis, design, implementation, testing, installation, deployment and maintenance*.

The analysis phase consists of preliminary analysis and system analysis i.e. requirements definition. In preliminary analysis, the organization’s objectives and nature and scope of the problem under study are defined. In system analysis, project goals are defined into the intended function and operation of the application. In this process, facts are gathered and interpreted, as a result problems are diagnosed and improvements to the system are recommended [11].

Desired features and operations are described in detail in the system design phase. This includes the description of screen layouts, business rules, process diagrams, *pseudocode* and other documents [11]. In the development phase, executable code is written. The integration and testing phase brings all the parts together into a unified testing environment. In this phase, the software is put into production. The final stage of initial development is installation and deployment. In this stage, the software is put into production within the business. During the maintenance phase, the system is assessed to ensure that it is not becoming obsolete. Changes are made to initial software and it is evaluated continuously in terms of its performance during the maintenance phase [12, 13].

This approach to software development has advantages, in that it provides an opportunity for completely new design and development practices to be put into place, meaning that a Greenfield project is not constrained by prior work. However, this approach introduces disadvantages, since it restricts the opportunity to build on, and reuse, existing software systems, requiring all components to be written anew. Therefore, this makes this approach more time-consuming and, as a result, potentially more expensive for companies to adopt [14-16].

In contrast, an approach in which previously developed software systems may be reused or modified to meet new requirements, to be used as an element of a new system, is called the extractive approach [17]. Component-Based Software Development (CBSD), an example of the extractive approach, is a model for building software systems or applications from *software components*, where a component is defined as "*a reusable software product which is independently developed and deployed to perform specific functions with well-defined interfaces*" [18]. An interface is "*a service abstraction that defines the operations that the service supports, independently from any particular implementation*" [19].

In CBSD [20], the set of activities involved in building a system consists of: *component assessment, component adaptation, requirements analysis, design, implementation, integration and testing*. However, due to extensive use of existing components, the realisation of CBSD development activities are different to traditional approches. CBSD requires focus on not only system specifications and development but also requires additional consideration to overall system context, individual components properties, and component acquisition and integration process [21].

CBSD is divided into component development and system development phases. The system-level phase places emphasis on finding and verifying proper components; the component-level phase emphasises reusable design. CBSD builds systems as an assembly of components which are themselves developed as reusable entities. Thereafter, the system is maintained by customising and replacing such components [21].

The component assessment phase is responsible for finding and evaluating the components. The requirements analysis phase involves the assessment of suitable and available components and defines a complete behavioural description of the system to be developed, including the functional requirements and non-functional requirements; such as aspects of performance, software maintainability, or security, of the system. After the requirements analysis has been completed, the design phase commences, including problem-solving activities and planning for the new system. The implementation phase places a preference on the selection of available components over the development of new components, including the modification of existing available components according to new requirements and design specifications. The integration phase includes component composition, and the testing phase is responsible for component certification [22].

An advantage of CBSD is that it reduces development time since it requires less time to buy a component than to build, design and test a new one. Moreover, it increases flexibility as CBSD systems allow implementation of components and therefore, there is more choice of component to be selected to meet a requirement. Reusing a component that has been already tested in many different applications will enhance the quality of the system. The maintenance process for CBSD is much lower when compared to Greenfield since the obsolete components can be easily replaced with new enhance ones [21]. Therefore, CBSD reduces the development cost, time-to-market and provides the customer with a system that is efficient in meeting the changes required by the customer.

One of the disadvantages with CBSD is the difficulty of requirement satisfaction. Component selection is an iterative process. The result of component selection depends on the success of its classification and retrieval mechanism, where a wide variety of component repositories are considered for performing component search. The component either might not be found or when found might not perform the specific function or fail to interoperate with one another [21]. Therefore, the requirement phase takes an important role in the overall development process. A good requirement analysis helps the user in the selection process. However, a poor requirement analysis can lead to component mismatch. The mismatch occurs when the retrieved component is not fully related to the user requirements. In other words, the mismatch may happen due to any incompleteness, inconsistencies or incompatibilities of user query with selected component [21].

Another disadvantage of CBSD is the location of suitable components. When software engineers include a component search as part of their normal development process they need to be reasonably confident of finding suitable components from repositories. Interoperability is another disadvantage of CBSD. This is a considerable challenge [21]. CBSD needs to ensure that component services are provided through standard interfaces to ensure interoperability. Moreover, there are challenges in unit and integration testing for CBSD. Unlike traditional applications such as Greenfield, individual components can be reused in a different set of applications. This complicates the integrated testing process of the overall system [21].

However, the advantages of CBSD are promising and the disadvantages only increase the need for careful preparation and planning to address these challenges by defining guidelines, standards and an open architecture for CBSD [21].

There are four factors that lead an organization to select CBSD over Greenfield software development approaches [22-24]:

- The challenge and cost of developing software from scratch and complexity associated with huge codes that are difficult if not impossible to understand. CBSD component reuse characteristics can simplify building new systems.
- The cost of maintenance of legacy software systems after they have been developed; the plug and play characteristic of CBSD allows the organization to replace components with new components without affecting the performance of the remainder of the software system
- Rapid development expectations. CBSD reuse components already developed instead of developing the overall system from beginning. This will reduce production time.
- Developing a quality product via supporting the construction of new software systems with validated, existing software components.

In this thesis, a framework is introduced in order to overcome the disadvantages of CBSD, assisting users in the component selection process. We demonstrate the usefulness of this framework through the development of a prototype implementation and experimentation, showing that our approach reduces the cost and complexity of analysing non-functional requirements in component selection, including an analysis of qualitative user preference.

1.1 Component Selection

In order to overcome the challenge of selecting the right component in the CBSD component selection process, components must be well documented. Each component has a file associated with it for providing a detailed description of its capabilities, so-called *Component Description*. Component descriptions facilitate the selection and retrieval of components, which are deposited in storage called a *Repository*.

One of the problems in actualising software reusability in organizations is the lack of means to locate and retrieve existing software components [25] There might be software available for use for a new application in the repository, but it is difficult for developers to locate the software or even be aware that it exists [26]. Moreover, it is possible to mistakenly retrieve a component which does not match the requirements [27].

These issues cause existing components to be re-invented over and over again [28]. The costs of modification adds to overall project expenses [29]. Software reuse is deemed as a key component of software productivity and quality by software engineering research [30, 31]. Effective software reuse requires that the system developers are able to find components that can be used in conjunction with other components to form larger units [25]. Thus, an essential stage for a low-cost and high quality software reuse is to provide the means for organizations to quickly search a repository. The prerequisite for this is an automated mechanism for component selection.

Component selection is typically based on the services provided by components (i.e. the functional requirements supported by the component) and its codified interface. While current component-based technologies successfully manage these functional interfaces and this degree of selection, this is insufficient for our purposes, as it is necessary to also consider and select components based on their non-functional requirements, i.e. performance, resource usage, etc; use cases, and test cases [23, 32]. Accordingly, organizations must also consider the extra dimensions of non-functional capabilities that help them to meet quality needs and client expectations.[33].

1.2 Motivation

Building large systems by integrating and reusing components is a viable development strategy for software companies [34]. However, there are several disadvantages identified in the reuse of existing components including the lack of clarity of component description (i.e. description complexity) and the difficulty in finding the most closely related component, (i.e. component mismatch), which is the result of an ill-defined component descriptions [35], and inadequate interpretations. A developer needs knowledge of both functional and non-functional requirements to describe a component or interpret a component description.

The software development industry has an expectation of quality software. In other words, it has implicit expectations about how well the software will work [36]. These characteristics include how easy the software is to use, how quickly it executes, how reliable it is, and how well it behaves when unexpected conditions arise [36]. The concept of quality is also fundamental and essential to software engineering, and to develop a quality software system, both functional and non-functional characteristics must be taken into consideration [37].

There are many approaches [6, 32, 33, 38] that can provide the necessary functional and non-functional knowledge to developers in order to improve the accuracy of component retrieval. These approaches utilise manual [33] or semi-automated techniques, such as brainstorming, survey–structured interviews and discussion methods.

Interviews are a traditional method and provide a quick way to collect large amounts of user preference and requirements data. A structured interview based on requirements analysis questions is sufficient for selecting components in many cases, however their effectiveness depends on the quality of interaction between the interviewees and interviewers. Thus, the result of the interview depends on the skills of the interviewer and the question design [39].

Informal discussions are a commonly used approach, and often become the default method for component selection. These discussions involve all of the different stakeholders who may participate in the development. However, due to the number of people who may be required to attend, these sessions can be difficult to organise and manage. Further, the success of discussion depends on the participants' experience, expertise and understanding of their non-functional requirements [39].

Another approach commonly used is brainstorming, which is an informal discussion to rapidly generate as many ideas as possible without focusing on any one area in particular. One of the advantages of brainstorming is that it allows the discovery of new and innovative solutions to existing challenges. However, the disadvantage in using brainstorming is that it does not explore the description in detail [40].

These techniques are sufficient to interpret and understand high-level component descriptions; however, they are typically time consuming and, accordingly, costly. This is at odds with the purpose of employing CBSD, which is designed to reduce time and monetary cost in development. Most of them require a decision maker to define processes and criteria templates. These studies [6, 32, 33, 38] are interpreting the relationships between quality attributes throughout the system development and usage.

This thesis considers non-functional requirements due to their importance to the success of software systems. In order to successfully select the right component, a detailed understanding of non-functional requirements is required, both in defining user queries for suitable components, and in interpreting them as part of the selection process. This requires that the task of *defining non-functional requirements* be explored in more detail.

A framework has been proposed that provides the necessary tools and techniques for automating the process of component identification and selection base on non-functional requirements. An automated process for this purpose requires the design and development of appropriate tools and techniques for automatically analysing the user query and the component descriptions in terms of their non-functional requirements for the purpose of component assessment and finally component matching. Moreover, the framework is able to automate the processes of query improvement, component assessment and component matching.

To add non-functional requirements to the process of analysis, we must introduce a mechanism for defining such requirements in a way that they can be understood and analysed by automated tools, along with requirements' interrelationships. Requirements might be directly or indirectly proportional to each other; for example, *usability* as an attribute is directly related to the *effectiveness* of a component but since *effectiveness* has *functional suitability* as a prerequisite, therefore, *usability* has an indirect relationship with *functional suitability*. Moreover, the interrelationship of one requirement might subsume, prevent or contribute to the fulfilment of another [41].

One of the principal approaches to defining non-functional requirements is the product-oriented approach [38], which aims to develop formal definitions of non-functional requirements. This approach helps a software system to be quantitatively evaluated by the degree to which it meets its requirements [42]. In addition, to correctly understanding non-functional requirements and their impact on the specific software system, the relationship between requirements must be analysed for conflicts or potential for trade-off..

Within this approach, non-functional requirements need to be specified formally by stakeholders, however, this is often not a natural process for stakeholders, with non-functional requirements typically remaining hidden in designs and related documentation. Consequently, current approaches seek the assistance of experts in non-functional requirements and requirements engineering techniques [41] to manually analyse and extract formal definitions of requirements [43, 44]. A non-functional requirements expert is a person who has knowledge of how non-functional requirements impact the quality of projects, how non-functional requirements can be combined and how to measure their metrics.

As an example, when building airport traffic software, one needs to hire airport traffic experts and conduct many discussions with themin order to avoid any conceptual errors. An expert integrates the air traffic knowledge and requirements engineering techniques to formalise the stakeholder’s need. This approach, while producing the formal requirements descriptions needed, is both time consuming and costly.

Our approach builds on elements of requirements engineering, in terms of formal approaches to interpreting the characteristics of the component and building a comprehensive query. Our approach to automating this process is based on the development and use of an *ontology* for non-functional requirements, providing a formal way for specifying non-functional requirements, and their relationships, that can be reasoned about in an automated manner, supporting the reasoning, organising and representation of this complex information source [42, 43].

1.3 Problem Statement & Research Question

This thesis seeks to demonstrate a framework for the automated analysis and selection of components based on non-functional requirements, building on the development of an ontology for non-functional requirements.

To achieve the above objective, the following research questions are considered:

1. What type of component description express all of the information that is needed by the developers' community?
2. Is this approach useful for the developers' community?
3. What is the efficiency of the approach?
4. What does the developers' community understand non-functional requirements to be?

1.4 Contributions

Component-Based Software Development (CBSD) facilitates software development in terms of quality, time and cost of development. Component retrieval approaches are one of the main processes of CBSD, designed to identify the most relevant component based on the user query. However, the current retrieval approaches do not work efficiently in finding relevant components according to the users [45]. The approaches are not strong in non-functional requirements analysis, and therefore do not meet overall software development needs.

Our main contribution is a framework for the automated analysis and selection of components based on non-functional requirements, building on the development of an ontology for non-functional requirements. As users are not familiar with non-functional requirements, any approach must produce and use non-functional requirements knowledge in support of the selection process in a manner that is easily understandable and usable by stakeholders. A set of non-functional requirements are defined based on analysis of industry studies [6, 20] and the literature [46-51], including their relationships and interdependencies. This study shows how the proposed ontology and framework can be used to better support timely and informed selection of components.

1.5 Thesis Structure

Chapter 2 discusses background information pertinent to the research discussed in this study. In order to provide a basis for the decisions made in implementation, this thesis provides a literature review in chapter 3 to encompass relevant work in the area and to explicitly define terms that will be used throughout the remainder of the document. There follows a discussion of the relevant technology of particular interest such as ontologies. Chapter 4 presents a discussion on the challenges that occur when developing component-based software. Chapter 5 defines the methodology used to solve the outlined problem and concentrates on the improvement of existing work. Chapter 6 details the implementation and design of the component descriptions, ontology and algorithm. These experiments are carried out to verify that the approach is valid and to also provide runtime statistics to show the effectiveness of the solution. The result of the experiment, in conjunction with the experimental framework, forms the basis of chapter 7. Finally, in chapter 8, conclusions are drawn from the theoretical and practical work to state what impact this has on the field. Furethermore, future work, as discussed and addresses other possibilities for this approach.

2 Background

2.1 Introduction

This chapter provides background information required for the development of the new ideas and experimentation described later in this thesis. This chapter begins with a discussion of necessary background concepts required to understand the processes involved in component identification and retrieval in CBSD. Moreover, The principal components of CBSD are defined.

To begin this chapter, Component description types such as non-functional and functional are disccused. Then, All existing ways of defining functional or non-functional capabilities of components as component descriptions are discussed. Additionally, All techniques that are used for retrieving the components with discussed type of description plus introducing a technique to assess the retrieval technique performance are covered. Finally, The importance of non-functional requirements is introduced and discussed information required to build a non-functional requirement ontology.

2.2 Component Retrieval

As we noted earlier in Chapter 1, the CBSD process encourages the use of existing software components. The components required within CBSD are identified by analysing their formal descriptions, which are typically based on a combination of product documentation, client information, and the previous experience of the developers [35]. *A component description* describes the capabilities of a component. There are two types of component descriptions for a product: functional, those that describe what the component does and non-functional, those that define the quality attributes of the component.

Typically, there are four methods for defining component descriptions, although natural language is the most commonly adopted method [52-54]:

- Natural language-based [55]
- Behaviour-based [55]
- Signature-based [56]
- Formal specification-based [57]

Natural language descriptions classify software using *facets*, where each facet represents a keyword and describes a characteristic of a component. For example, software components can be described by the functional requirements, non-functional requirements, and their implementation details. These fall naturally into facets that can be ordered by their relevance to reusability [58]. A drawback of the facets is the lack of concepts to describe links (relations) between facets, terms or components classified in the schema. Although it is possible to define the term of an existing facet as a new facet, it is not explicitly declared as a subtype relation or specialisation [59] and also it usually does not cover non-functional requirements due to their complexity. Moreover, due to the subjective nature and vagaries of natural language, different developers may provide different descriptions for the same component, introducing complexity in component matching. Accordingly, the description of a component can be sometimes ambiguous, confusing or even contradictory. Furthermore, most of the time the component documentation is incomplete [52-54].

Behavioural-based descriptions are limited to functional behaviour of components. In order to describe a component, this method uses a set of input-output pairs, which can be viewed as an approximate specification or description of the code component [60]. For example, consider a component library that contains a number of string-handling function components. Suppose that the functions are described by the sample input-output description, where each sample consists of the actual arguments before execution, the argument after execution and the return value after execution. An as example, a component “sconcat” takes two string as arguments then concatenate the second string at the end of first one. Its sample description might be: [(“abc”,”xyz”), (“abcxyz”,”xyz”), (“abcxyz”)]. For developing this type of description, the user records the set of inputs, which produce same output; input/output checks can validate the functionality of component, however the non-functional requirements can not be validated in this method [61].

Signature-based description uses signature information derived from the component. This description is used for matching at both function and the module levels. The signature of a function is simply its type and signature of a module is both a multi-set of user-defined types and a multi-set of function signatures. For example if a user is looking for a specific function, a query is performed based on the function’s type, the query includes a list of types of its input and output parameters. Using this method is insufficient as it also does not consider non-functional requirements when selecting a component.

Formal component description uses mathematical or formal languages to describe the deposited component description and user requirements. These descriptions include pre- and post-conditions of the component. For example, post conditions may state related properties of returned values. Similarly, preconditions of related functions may state related boundary conditions of input values [62]. A formal method is more powerful than other methods because they precisely record the system’s functionality, both expected and delivered [63, 64]. However, this method can be very costly and time-consuming when there is a project with many dependencies (i.e. relationships between components) [63].

In order to re-use components deposited in a repository, there are some techniques, so-called *retrieval techniques*, which facilitate the retrieval of the components. There are six general types of component retrieval techniques[65]:

- Browsing-based retrieval technique,
- Behavioural-based retrieval technique,
- Specification-based retrieval technique,
- Signature-based retrieval technique,
- Query-based retrieval technique,
- Semantic-based retrieval technique,
- Internal retrieval technique, and
- External retrieval technique,

Browsing-based retrieval facilitates the user request by representing the component relationships as a weighted graph, which helps the user to navigate and return the required component [65, 66]. In this method, the user is able to browse components by their names, traversing the use-relations (the nodes of the graph correspond to components and the edges linking the nodes correspond to cross component usage), searching similar components that clustred to one node. The nodes in the graph are ranked by their weights. Component relations might be constructed through the use of concept keywords or a component's functional properties. Each component property represents a node within the graph; a relation is in the form of a logical formula, based on the specifications (interface's pre- and post-conditions, syntactic description of the interface, or specification of operations in the interface [67]). Browsing-based retrieval uses different relations to build a suitable index for the component. This approach needs a navigation structure, which requires a formal concept analysis that feeds a hierarchical collection of concepts to navigate [68, 69]. In addition, the formed concept analysis should also investigate the non-functional requirements to identify the most relevant nodes.

Approaches to behavioural-based retrieval [70-73] are based on the execution of software components. As mentioned previously, the user executes the component with their set of inputs and if the returned output of execution is relevant to user expectation, then the selected component is the one that suits the user requirements. These types of retrieval consider only functional requirements.

Approaches to specification-based retrieval [57, 62, 67, 74-76] use formal language and mathematical logic to describe the desired component. A formal language is constructed mathematically using a combination of formal rules and grammars, and may be domain-specific. An automated theorem prover is used to check the validity of the formula. Automated theorem proving is responsible for proving mathematical theorems with the help of a computer programs. This technique may be time-consuming and difficult with respect to practical applications [68]. The retrieval process commences by calling the *theorem prover* method for each and every component in a repository till a match is found through an automated theorem prover validation. Each component is associated with a formal specification that captures its relevant behaviour. Any desired relation between two components (e.g., refinement, matching, or reusability) is expressed by a logical formula composed from the associated specifications.

Signature-based retrieval compares the signature of repository components with a provided query signature. The signature of a component consists of the component method name; method arguments, arguments type, and method return value. There may be some components that do not exactly match but the signatures are somehow similar. Those components are also considered a match due to the user ability to modify the component or query[77]. This technique considers only functional requirements.

Approaches to query-based retrieval formulate a user request in the form of a query to select a number of components as a result [65].The user sends the requirements by using natural language, requiring a translation process to SQL using a query interface, and building on the following processes:

- Natural language processing: facilitating the analysis of text using automated approaches based on both a set of rules and a set of technologies [78].
- Ontology development: the specification of a domain, typically describing the hierarchy of concepts and relating each concept's properties with it.
- Domain modelling: providing a conceptual framework of the things, which are concepts of real-world in the problem space.

The above processes improve the result of query-based retrieval techniques. However query formation remains a difficult task, as a user often does not have a clear understanding of how to search and what is needed. [79].

User queries are mainly specified based on the understanding of functional requirements, which are decomposed into specific processes and actions that are encapsulated in the object according to the domain model. The component objects are also analysed and the matching process is responsible for retrieving the components with the highest percentage of similar processes and actions [80]. The result of all these steps is a semantic query construction but some approaches [81] consider semantic component descriptions as well. A semantic query consist of more meaningful terms, which are chosen from domain model [80]. In the same way, a semantic component description considers the lexical relationships among terms chosen to describe services and operations that are provided by a component [81].

As an example, non-functional requirement based queries look for a component using a set of non-functional requirements in a format of names paired with values. Each non-functional requirement name represents the desired quality that the component should have and the value indicates the degree desired for that component quality. The following format is a simple query for a component that is 100% reliable and secure.

" Reliability ":"100", " Security ":"100"

As mentioned in query based technique, semantic-based techniques use a domain ontology to provide semantics in order to refine user queries. This supports matching between a query, articulated in natural language, and component descriptions, which are stored in a repository describing a set of reusable components [80, 81].

The internal techniques use information retrieval thechniques [53] to extract some identifiers such as comments, classes, attributes, methods and parameters of the component. Then, they separate and normalise the identifiers and, finally, index them. The underlying assumption is that programmers use meaningful names for code items (identifiers). When this technique is used, there is no external representation of a component and the component can be considered as a particular type of document [82]. The internal techniques' inputs (identifiers) are not useful for non-functional requirements based selection.

The external techniques focus on an external description associated with the software. Natural language-based descriptions [53] and facets [54] mentioned in Chapter 1 are among the category of external component description. These techniques, index the component based on its external representation [65].

All of the retrieval techniques aim to have high-performance measures. The following models are used to assess the performance of retrieval techniques [83].

Searching effectiveness refers to how well a given method supports finding relevant items in a database. The items that satisfy software requirements are relevant. We have evaluated searching effectiveness with recall and precision, the traditional measures.

Precision (P) is the number of relevant items (r) retrieved over the total number of items (T) retrieved. Precision is defined as:

$$P = \frac{r}{T}$$

Recall (R) is the number of relevant items retrieved (r) over the number of relevant items in the database (TC). Recall is a performance metric, which measures the proportion of relevant material that is retrieved. Recall is defined as [65, 84]:

$$R = \frac{r}{TC}$$

The closer these P and R values are to 1 identifies better performance of the retrieval.

2.3 Requirement Engineering

In the area of requirements engineering, there are two areas for which ontologies are developed [85, 86]: functional and non-functional requirements [87].

Etymologists [88] define ontology as the knowledge of beings, and all that relates to being. The term “entity” is used to describe all things which “are” and exist. According to this point of view, stones, animals or people are “entities”. Mathematical objects, even those which are only imagined, are also considered beings. Science and knowledge refer to, or examine, a type of entity as either physical, (e.g. in the physical sciences) or abstract, as in mathematics and the vast majority of the computational sciences [38].

Ontology is an explicit formal specification that represents the entities that exist in a given domain of interest and also defines the relationships that hold among them. Ontology provides a machine-readable description of content and capabilities of the domain. Ontology is necessary in order to be able to reuse or share the knowledge multiple times in different applications [89].

More information is provided by ontology (reasoning process) using a program called a *reasoner [8*9*]*. Logical consequences (knowledge) are inferred from sets of the ontology's logic and user-defined rules (axioms) using a reasoner [90]. An inference engine is need for this process, which works based on its rules. An ontology language (such as OWL [91]), and a description language are used in order to specify these rules [92, 93] .

Functional requirement ontologies include the most general concepts needed for any domain due to the nature of functional requirements. Functional requirements are features of the developing software system. A functional requirement ontology can be used to store the domain knowledge of software requirements. For example, for representing functional requirements in ontology, the objects may become functions in software and attributes may be the scenarios of using the functions such as user (who), time (when), method (how). Moreover, relations such as complement, inherit and contradict may exist among functions [94].

In contrast, a non-functional requirements ontology specifies a conceptualization of the non-functional requirements domain in terms of concepts (i.e., general non-functional requirements), sub-concepts (i.e., more specific non-functional requirements) and relations. Concept or sub-concepts can be associated with their instance, which form the component descriptions [95]. Non-functional requirements ontology is necessary to facilitate the analysis of non-functional requirements due to following reasons: non-functional requirements are always related to specific domains and also affected by context. Therefore, it is difficult to ask users to provide non-functional requirements explicitly. As a result, a significant portion of non-functional requirements are neglected [96]. Moreover, requirements analysts usually lack a deep understanding of relevant quality requirements of an application domain, thus additional knowledge support is needed in the process of asking the right question to elicit requirements. Furthermore, quality requirements are stated informally and there are few approaches that define a quality model or attach metrics to non-functional requirements i.e. qualitative or quantitative measures of the requirements [95].

Therefore, as a result of having an ontology in place, there is a requirement to have a taxonomy and an appropriate structure for non-functional requirements interrelationships and definitions.

2.4 Quality Model

Non-functional requirements are typically categorized by quality models. These models provide general definitions, which facilitates understanding of non-functional requirements. Moreover, they can be considered as a good reference for software engineering practices such as quality measurment of software designs with software quality's characteristics, sub-characteristics and their metrics, which they provide [97].

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) published 25051 [46], 25023 [50], 25010 [49], 25020 [47], 25022 [51] and 25021 [48] standards for defining requirements, evaluating software products, measuring quality aspects. These standards are generic and define quality measures, quality requirements for components, quality in use and provide a measurement reference model.

The taxonomy used for our ontology is based on the quality model ISO/IEC 25010, which is published to introduce general software characteristics, with each characteristic divided into sub-characteristics. This model has two categories of quality: the *quality in use* model, which has five characteristics and the *product quality* model, which has eight characteristics. These models' characteristics also have a set of related sub-characteristics as described above (Section 2.4). A taxonomy is a structured overview of classes (characteristics), subclasses (sub-characteristics) and instances of a domain [98]. A sample of the taxonomy is defined in Chapter 5.

A non-functional requirements ontology is a formal description of non-functional requirements and the relationships between them [99]. It facilitates non-functional requirements knowledge sharing and reuse [100]. It defines a common vocabulary for researchers who need to share information in the non-functional requirements domain. It includes machine-interpretable definitions of basic concepts in the non-functional requirements domain and relations among these concepts [101]. The analysis of non-functional requirements definitions is needed to discover the relations among them. These relations are necessary for constructing an ontology.

The first two chapters, discuss the importance of non-functional requirements and the challenge of analysing them. Moreover, the necessary tool and techniques to analyse and elicit them is discussed. In the next chapter, a review of related works is presented.

3 Related work

This chapter reviews the existing work on component selection based on non-functional requirements, with emphasis on requirements analysis considering non-functional requirements. These existing works employ the techniques and tools that are introduced in the previous chapters. We discuss how the existing works have addressed the concerns raised regarding analysis of non-functional requirements in Chapter 1. The chapter concludes with a summary of the perceived shortfalls and gaps in the current literature.

In this chapter, existing works that utilise software requirement techniques to elicit functional and non-functional requirements and their relations in order to produce requirements knowledge for component selection is discussed. Specifically, requirements' categories, requirements definitions and requirements' relationships are a focus. Both (1) human or manual approaches, and (2) automated approaches, commencing with a discussion of manual approaches are considered.

3.1 System Requirements through Goal-Orientation

Chung et al. [7] propose a goal-orientation (requirement-orientation) approach, where system requirements are represented as goals. In this approach, goals are identified as a higher-level (general) form, that are decomposed into sub-goals (specific form). The hierarchy of goals and sub-goals then represents the identified requirements. Goals consist of non-functional requirements (soft goals) and functional requirements (hard goals). Moreover, these goals might be from different stockholder with conflict or be synergistic with each other. For example, one group of stakeholders may say "the security of the system should be of the utmost concern," while at the same time another group may say "the system should be as easily changeable as possible, and that should be the top priority" [7]. One of the important steps of this study is eliciting, analysing, correcting, and validating the hierarchy of goals by asking what, how, and who questions repeatedly.

Chung et al. [7] apply a set of techniques to elicit the set of requirements, including interviews, checklists, and a review of the existing documentation of a system. Their approach relies on manual reasoning to select the best set of non-functional requirements. The seletion is based on the interrelationships whereby non-functional requirements subsume, prevent or contribute to the fulfilment of another non-functional. For example, consider two non-functional requirements being “the system should be easy to use” and “have a moderate cost”. The interrelationship between these two goals is defined as negative, which means that making the system easy to use prevents developing the system at a moderate cost [7, 8, 37]. The Chung et al. framework can distinguish among parent (general) or children (specific) forms of non-functional requirements and also their relationships; however, it requires a significant investment of human and financial resources due to the manual nature of their approach, introducing inefficiencies. Further, the use of manual reasoning introduces opportunities for human error.

3.2 Component Selection using Quality Models

There are several other manual approaches, based on the development of quality models that are of interest. Franch et al. [9] introduce an approach to build a quality model based on ISO 9126-1[102] to facilitate the description of requirements and components, where selection is based on negotiation between user requirements and components capabilities. This work is selected in order to investigate their catalogue of non-functional requirements and their relations. The initial step of their approach is analysis and description of a domain of interest. They illustrat this step by providing an example of a mail server. Then, the outcome is used to build an ISO-based quality model. They tailor the ISO model either by adding or eliminating sub-characteristics to or from characteristic/sub-characteristics or modifying qualities' definitions. For example, they add basic and advanced suitability to the ISO quality characteristic. The tailoring stops when the entire hierarchy of elements can be directly measurable. Moreover, they identify three types of relationships for their model, which assist in reasoning about non-functional requirements [9]. These relationships are collaboration, damage and dependency relations. Requirement A has collaboration with requirement B when growing A implies growing B. On the other hand, A damages B if growing A implies decreasing B. Finally, some B conditions should be fulfilled before A completion.

This approach partial definitions make the automation of non-functional requirements' management difficult. Since the metrics and quality attributes' relations are based on a specific domain or context, their model cannot be used as a good reference for domain independent component selection, which is necessary for automation [23, 80].

Gemma et al. [103] have extended the approach introduced by Franch et al. by providing a software system for selection of components. They define a set of tools introducing:

- Quality model tool: defines software quality factors, facilitates reuse of them, states relationships among them, assigns relationships among them and facilitates analysing of requirements.
- Evaluation tool: facilitates the evaluation of candidate components using a quality model.
- Component selection tool: responsible for matching user requirements against components specifications.
- Taxonomy tool: facilitates sharing or reuse of quality models.

Gemma et al. did not automate their process of component selection. However, they have developed a semi-automated framework that minimises the need for human actors, requiring a:

- *domain expert*, who analyses the domain in order to build the quality model, and a
- *requirements engineer*, who defines requirements in order to perform the selection of a component.

Joerg et al. [6] propose an experience-based method to elicit, analyse and document the non-functional requirements using checklists and a prioritization questionnaire. The process starts with the *requirements engineer*, who tries to identify the priority of non-functional requirements by interviewing the developers. The next step is the non-functional requirements elicitation, where the quality model is explained to developers and analysed by the requirements engineer. The result of this step is the production of a checklist, which improves the non-functional requirements elicitation by providing additional experience and information. This step is very time-consuming [104]. Prioritization questionnaires facilitate the identification of the most important quality attributes in a specific context.

This approach uses a repository, which contains the functional and non-functional component descriptions extracted from the component marketing brochures. The matching process starts by comparing the software goals (requirements) with component descriptions. The selected component will be the one, which gains the highest-ranking score in terms of relationship and number of matched goals [7, 8, 37].

3.3 Ontology-base Selection of Components

Moving to automated approaches, firstly approaches that use ontology to select the components are studied, and then those that use ontologies to select components based on non-functional requirements. Sugumaran et al.'s [80] approach is based on the use of an ontology to select components, which are in form of objectives, processes, actions, actors, and objects. This format is specified for their domain model in order to provide the context information. The ontology makes sure the use of appropriate terms in users' query based on an application domain (an auction application). To select components, a developer specifies a query in natural language. Then, ontology is used to evaluate and revise the user query. To select a component, the user query is decomposed into specific methods according to the domain of application, and compared to the methods of components stored in the repository. The selected component is the one that attains the desirable percentage of supported methods [80]. Sugumaran et al.'s approach provides the benefit of an ontology that captures specific domain knowledge and then provides that knowledge during the selection process, but this ontology cannot be used for other domains since it is designed only for a specific domain such as the auction domain.

Yao et al. [81] define a reusable component as a service and used WSDL [105] , which is an industry standard description language for service description to describe the components. This language is used to describe interface, data type, binding information and address information of components. They use web services to describe components. Web services are standardized mechanism to describe, locate and communicate with applications [106]. The process starts, when the user's query sends to system in natural language. Then the query is translated into a semantic representation format. The approach employs a domain ontology to refine user queries and to match a user query agaist components in a reusable repository [81]. Yao et al's approach is based on Sugumaran et al. ontology, with their ontology designed for a specific domain and the ontology definitions limited to the specified project domain (context). Moreover, there is not a clear definition of their ontology in the article. Therefore, their ontology is not compatible with other domains. Furthermore, they treat a software component description as a description of the services provided by the component. Since services describe the functionality of the components, this approach selection is based on functional requirements.

3.3.1 Ontology for Non-functional Requirements

This Section provides a review of existing work that uses ontologies to select components based on non-functional requirements. Li et al. [107] build an approach to retrieve components using a non-functional requirements ontology-based approach. Each non-functional requirement is specified with a set of keywords and concepts in the ontology. Keywords consist of basic component information such as the name of the component, environment information such as required resources for using the component, and functional capabilities of the component. The ontology is designed in a way that can facilitate the production of components' specifications in the repository.

To select a component, the user sends a natural language query to the system then the system identifies the proper keywords and finally validates them using the ontology. Moreover, This approach proposes an algorithm, which helps to calculate the relevance of a component to the user query (keywords). The selected component is the one that gains the highest percentage of the relevance [107]. The limitation of Li et al.'s approach is the proposed ontology. The ontology is actually a non-functional requirements taxonomy but not an ontology. It is only a hierarchical structure to classify the requirements while an ontology should be a hierarchical structure, which additionally defines properties (and their restrictions) and relationships between concepts explicitly [108]. Their ontology only has an “is-a” relationship, which is essential for classifying the requirements. They need to have their concepts (class) expressions, which provide the necessary conditions for the reasoning about functionality.

3.4 Summary

In the selection process, functional requirements are generally considered. However, users do not typically have enough knowledge of the development processes to also describe the necessary non-functional requirements. There are currently several approaches [33, 40, 43, 86], which apply manual techniques such as questionnaire and group interview to gather non-functional requirements knowledge. While these techniques can indeed elicit the necessary requirements, they are time consuming and prone to human error, increasing the cost and time of development. Automated approaches introduce the potential for considerable time and cost improvements for the software development process.

Across all quality-based component selection methods is the common concern over the required investment of time, and the commensurate cost to elicit the knowledge of domains, components and requirements. In the context of component selection, automated approaches to component selection can reduce cost and human effort, improving accuracy and enabling the full potential of component-based development. However, automated support for component selection is still in its infancy, due to the requirement for complex automated decision support techniques [109].

This thesis presents the design and implementation of an approach to automated component selection through an ontology-based non-functional requirements repository system. The next chapter describes the challenges that must be overcome to develop component-based software.

4 Challenges in Developing Component-Based Software

A framework, forming an efficient mechanism for automating the process of component selection based on non-functional requirements, is introduced in this study. Here, the key challenges that are faced in component-based development is described. In later chapters, possible solutions and design decisions to overcome the stated problems is introduced. This chapter illustrates how developing component-based software is a complex task. Moreover, existing solutions for the problem and their weaknesses are discussed. Prior to applying the existing solutions the weaknesses need to be addressed and resolved.

Complexity of Requirements Description

To understand the stakeholders' requirements, the requirement engineering team needs to gather and document the required information about the desired system. Nowadays, software systems are large and complex [110]. Therefore, requirements documentation is a significant process, with requirements produced and frequently interpreted by people with different experience levels and backgrounds [111]. Accordingly, in some instances, the process of requirements description may result in misunderstanding of the overall system requirements, as they may be incomplete or ambiguous [112, 113].

Incomplete Component Descriptions

There are two means for understanding the requirements of software components. The first is by analysing the components design document; this method, however, this is not practical due to the lack of availability of documents, and being time-consuming to analyse [114]. The second method is by referring to and trusting the component descriptions, which are mostly in natural language even though the natural language-based descriptions are not precise [115]. Clearly, the component descriptions are typically based on product documentation and the previous experience of developers [35], and there is no common language for characterizing them.

Reuse of Definitions

Non-functional requirements can be defined in term of qualities, system properties, design constraints, behavioural properties, system attributes or services, which depend on the type of the project [116]. This thesis is based on the qualities' definitions, which define the relationship among non-functional requirements and their prerequisites. The current non-functional requirements definitions are highly coupled with their project or domain context. Thus, they might not be reusable for the projects with a different context.

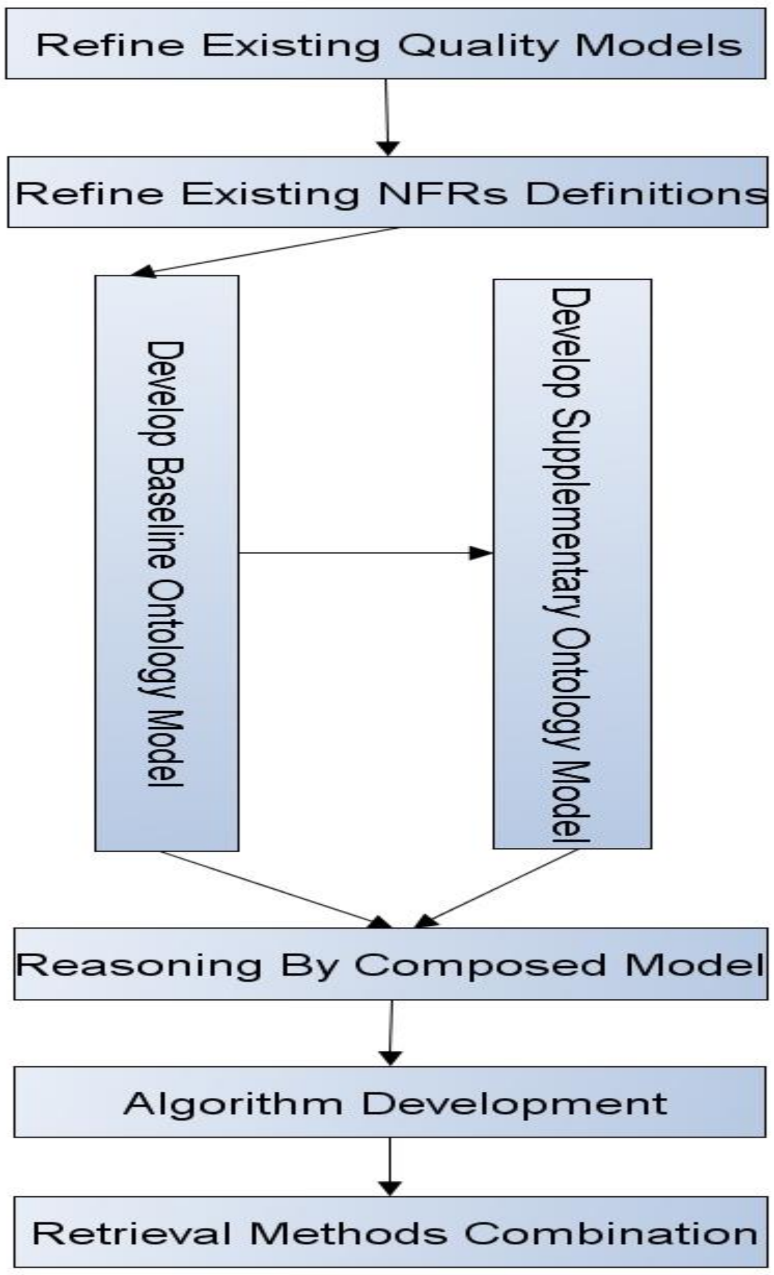
Complexity

Generally, describing non-functional requirements is complex as they can be treated as subjective i.e. they are influenced by personal feelings or interpretations. Moreover, they are related to other non-functional requirements and these relations will not necessarily be noticeable from a high-level view [32, 117]. A software engineer should develop a deep understanding of non-functional requirements relations and definitions to use them correctly in the project.

In this chapter, the major challenges facing component-based development occur when the emphasis is on non-functional requirements have been addressed. This leads to the design of a framework for automating the processes of component selection based on non-functional requirements.

5 Methodology

This chapter describes how this study has been conducted. In this study, to improve software reuse, an approach for component identification, starting with a search of potential components that match user requirements is presented. The component search is based on the component information available in an appropriate non-functional description format. A set of non-functional characteristics associated with a component forms its description. Components seach could be based on the functional characteristics of component. However, functional requirement examination is out of scope of this study. Two types of selection, namely, query-based retrieval and semantic-based retrieval are considered. Query-based retrieval techniques facilitate the formulation of the user request in the form of a query, which is based on non-functional characteristic of components. Semantic-based retrieval techniques facilitate the query construction with more meaningful terms (characteristics) that are selected with the knowledge provided by an ontology. Therefore, the ontology minimises complexity of analysing (described in Chapter 4) Non-Functional Requirements (NFRs) and provides enough information for matching the user query and a component description semantically [65]. An algorithm operates with the ontology to facilitate the identification and selection of components based on the non-functional knowledge described above. The Figure 5-1 shows the workflow of activities used to investigate the research problem in this study:



Refine Existing Quality Models
Refine Existing NFRs Definitions
Develop Baseline Ontology Model
Develop Supplementary Ontology Model
Reasoning By Composed Model
Algorithm Development
Retrieval Methods Combination

Figure 5-1 Thesis Workflow

5.1 Refine Existing Quality Models and Quality Definitions

In this study, non-functional requirements are selected based on an analysis of International Standard Organization (ISO) documents [46-51]. These documents contain sets of high-level concepts designed to facilitate software development. Moreover, they provide the required non-functional requirements definitions at a general level, which suits all type of project contexts. Pursuing ISO further, the hierarchical ISO model is a quality model in which the quality attributes are organized into a tree-like structure. The structure allows representing information using parent/child relationships: every non-functional requirement parent is connected with one or more non-functional requirements child. All non-functional requirements are not applicable for the use of CBSD.

5.2 Baseline Ontology Model

The baseline model is a conceptual model. The conceptual model is developed with the knowledge gained in first two steps of the thesis workflow (Figure 5-1): refining existing quality models, and refining existing definitions of non-functional requirements, providing the ability to interpret the quality attributes based on their properties such as degrees and relations. This knowledge builds based on non-functional requirements classes (concepts of ontology) and relations, where naming, definition of types, properties and relationships of entities are used to build knowledge. Based on the review and refinement of existing models, a set of relationships is identified and explained further in Chapter 6.

5.3 Supplementary Ontology Model

The supplementary model contains a set of baseline instances, which are instantiated from the non-functional requirement concepts defined in the baseline model. Each set forms the component description, which is deposited in the repository. The baseline model and a Table that defines the range of instances supply the necessary information, which is required for generating these sets. The brief design description of this model is presented in Chapter 6. A sample of the component description is also provided in Chapter 6.

5.4 Automate the Reasoning Based on the Main NFRs Ontology Model

Our ontology represents the non-functional requirements knowledge as a hierarchy of concepts, using a shared vocabulary, properties and interrelationships of those concepts in order to limit complexity and organize non-functional requirements information for the user. One of the key benefits of building an ontology-based approach is that the user can use a reasoner to derive additional knowledge about the concepts that are modelled. The reasoner highlights various relations among non-functional requirements. These relations, determine either the type of each non-functional requirement (general and specific) or its prerequisites.

5.5 Algorithm Development

The core of the approach advocated in this thesis is the development of an algorithm that operates with our ontology to extract non-functional requirements information. The input to the algorithm is a non-functional requirement name-value pair, which is provided by the user. The algorithm is designed so that it is able to search and elicit the non-functional requirement prerequisites from the ontology and then search for them in each component ontological description. The description format is Resource Description Framework (RDF) [118], which is a foundation to standardise the definition and use of resource descriptions [119]. Each description, with a complete list of prerequisites is validated and semantically matched to the user request. SPARQL [120] is the semantic query language employed for search parts where user inputs convert to SPARQL statements. The algorithm ranks the component descriptions with a partial list of prerequisites by using the weight assignment strategy. This strategy assigns a particular score to each relationship found in the NFR Component Description (CD). Each relationship score is based on a set of rules that precisely defines types of relationships. These rules are explained in details in the next chapter.

The CD with the highest score is the most relevant product to the user query (the score is used as the basic data to select a CD). Additionally, the algorithm is able to reduce the search area by eliminating the CDs that do not contain the requested Non-Functional Requirement (NFR).

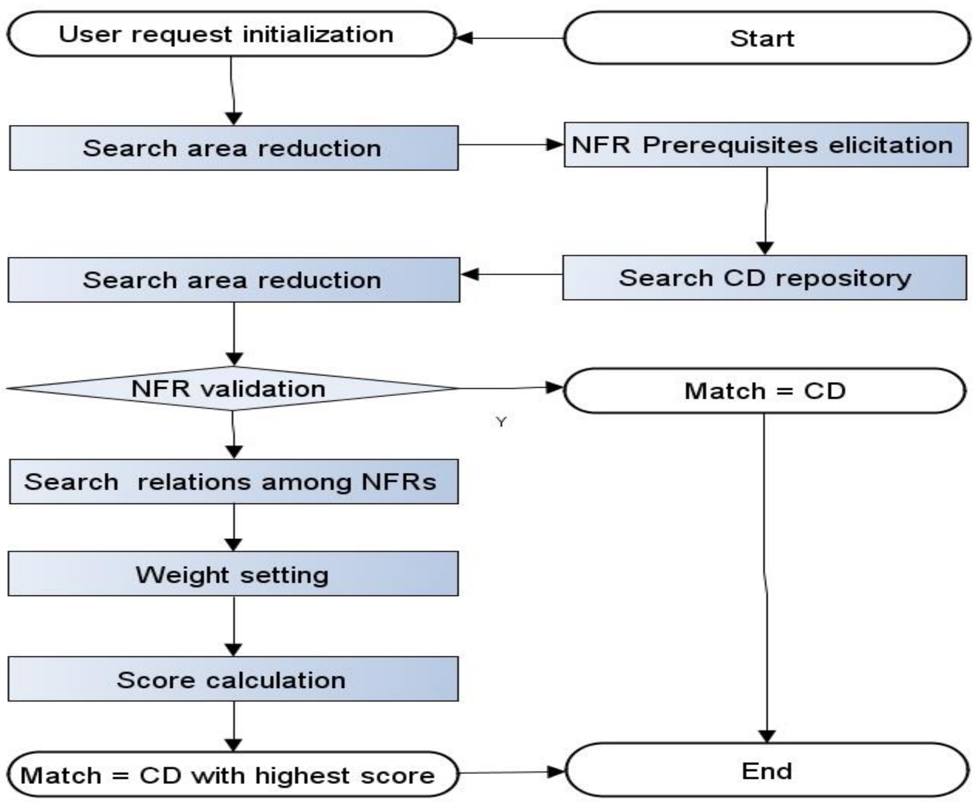


Figure 5-2 Weight Calculation Algorithm

5.6 Combination of Query and Semantic Approaches

Search algorithms for query-based retrieval are traditionally based on literal matching of keywords [121], such as, the non-functional requirement's names, to retrieve components descriptions. The performance is limited in this case since the conceptual non-functional requirement’s prerequisites are not applied. In order to select component with a semantic query, this study combines two of the retrieval approaches described in Chapter 2, which are: query-based retrieval and semantic-based retrieval. The retrieval approach is based on a semantic knowledge base in place of a keyword based index. Query and semantic-based approaches are based on an iterative retrieval process that is associated with an automatic query reformulation [65]. In this study, reformulation is defined as searching the repository against the requested quality attribute (non-functional requirement) and its prerequisites.

In this chapter, a high level overview of methods, used in the implementation and design of this study is provided. Two retrieval approaches, query based and semantic are combined, in order to map information found in component descriptions of a query into an ontology and to obtain more knowledge about the information. Two approaches are used for the validation porpuses. First, a qualitative end user evaluation with 30 participants. Last, a performance analysis that uses synthetic data to explore time to process a query. The following chapter discuss as the implementation process of this methodology.

6 Design and Implementation

This chapter discussion is divided into two main parts: design and implementation. A three-tier architectural model, which is the fundamental framework for the logical design is adopted. These tiers are the ontological model, query processing layer and data model. The design and implementation of the ontology is presented, along with a new algorithm for component identification and selection. Moreover, we have described the NFRs definitions for the purpose of this study. The final sections of the chapter focus on the overall strategy for implementation tasks and use cases (using UML).

6.1 Design

This Section describes the design structure of the prototype. The structure comprises layers, the properties of these layers and the relationship between them. The architectural Section is responsible for describing how these layers fit and work together.

6.1.1 Architecture

This study adopts a 3-tier software architecture consisting from an ontological model, a query processing layer and a data model, as shown in the Figure 6-1. The ontological model facilitates the ability to interpret NFRs and empower the user query. The middle layer (query processing) is responsible for all the communications between other tiers and the data model provides the necessary data for the required processes. This 3-tier software architecture supports independent change and evaluation of each tier. These are discussed in detail below.

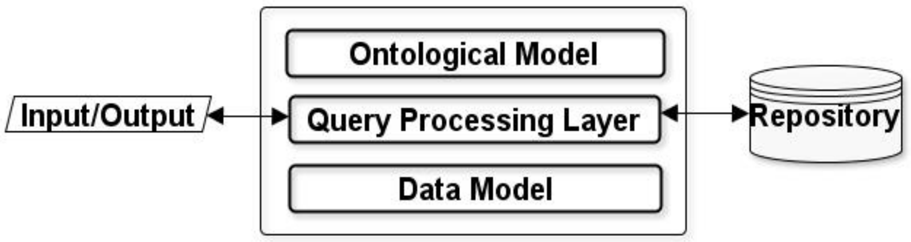


Figure 6-1 Architecture for Automated NFR reasoning

6.1.1.1 Ontological Model

The ontological model is responsible for capturing and reusing the knowledge of NFRs. This knowledge helps the application analyse the relationships among NFRs in the user query and component descriptions. The result of the analysis determines which component is valid and semantically matched to the user request. Each user request's element is assessed by the ontological model (as described in Chapter 5). The assessment mechanism checks for the query's element prerequisites in the query and, if a prerequisite is missing, then it will be added to the query. Moreover, the ontological model is used when the weighting strategy of algorithm needs the relationships among a description's NFRs.

6.1.1.2 Query Processing Layer

The query processing layer is responsible for all object interactions in the system based on its workflow and rules/logic. Moreover, it merges the ontology model with the data model (described in Section 6.1.1.3) to make a new model before processing of the ontological reasoning in runtime. The new model consists of classes (ontological concepts) and their ontological instances (data). The query processing layer manages the system rules to produce an output based on user input. Generally, this layer helps to answer queries over instances and ontology classes (OWL [91]).

6.1.1.3 Data Model

The data model is responsible for generating and storing data. It handles the transactions from query processing layer. This communication is performed by a query language to provide necessary data for other processes such as weighting calculation (explained in Section 5.5). In the weighting process a particular score is assigned to each relationship type (will be explained in Section 6.3.3) found in component descriptions. The component description with the highest score is the most relevant product to the user query. The score is used as the basic data to select a component description. Each description with a complete list of prerequisites will be valid and semantically matched to the user request.

6.1.2 Algorithm

An algorithm is the backbone of query processing layer, which provides an list of processes such as running queries, parsing descriptions and processing result sets. The algorithm is designed in such a way that it is able to elicit the NFR prerequisites from the ontology and search for them in each component description using the NFRs' knowledge. The algorithm is also able to rank the component descriptions with a partial list of prerequisites by using a weight assigning strategy. Additionally, the algorithm is able to reduce the search area by eliminating the component descriptions that do not contain the requested NFR.

6.1.3 Ontology

This study's ontology represents the NFRs knowledge as a hierarchy of classes (concepts), using a shared vocabulary, properties and interrelationships of these concepts. One of the key benefits of building an ontology-based approach is that it is possible to then use a reasoner to derive additional knowledge about the concepts that have been modelled. In this case, the reasoner highlights various relations among NFRs. These relations determine either the type of each NFR (general and specific) or its prerequisites. The ontology allows the NFR's prerequisites to be queried. Therefore, it is possible to formulate a natural language user query into a conceptual query. This change is to improve the quality of a query in term of the elements' (NFRs) relationships. The main purpose of introducing an ontology is to move from a query evaluation, based on words, to a query evaluation, based on semantic relations, thus moving from syntax to semantics interpretation.

6.1.4 Taxonomy

Taxonomy shows the NFRs' classification as a conceptual hierarchy. Therefore, it represents a variety of concepts and predicates. The taxonomy plays an important role in the conceptual modelling by providing substantial structural information. The taxonomy reflects a basic ontological structure with a clear semantics.

The Figure 6-2 is the taxonomy, which is the ontological model backbone. It is the tailored version of the ISO 25010 model explained in the previous chapter. This taxonomy consists of two types of nodes: parent and child. The Figure illustrates the semantic relations, i.e. "is a" relations between the parent and child node. A child node is a node extending the previous node. For example, 'reliability' is a child of 'product quality'. Association of each child to its parent is with an *is-a* relation and is shown with an arrow from child to parent like the arrow from 'trust' to 'satisfaction'. Moreover, there are five levels of nodes in the hierarchy. 'Thing' is the top most level of the hierarchy and the main parent that has 'quality attributes' as its only child. The second level of the hierarchy is 'quality attributes' which has two children: 'quality in use' and 'product quality'. The latter mentioned children are at the third level of hierarchy. The rest of NFRs can be considered the children of these two nodes and are in the fourth level of hierarchy. All the nodes groupings are based on the ISO/IEC 25010 standard's groupings.

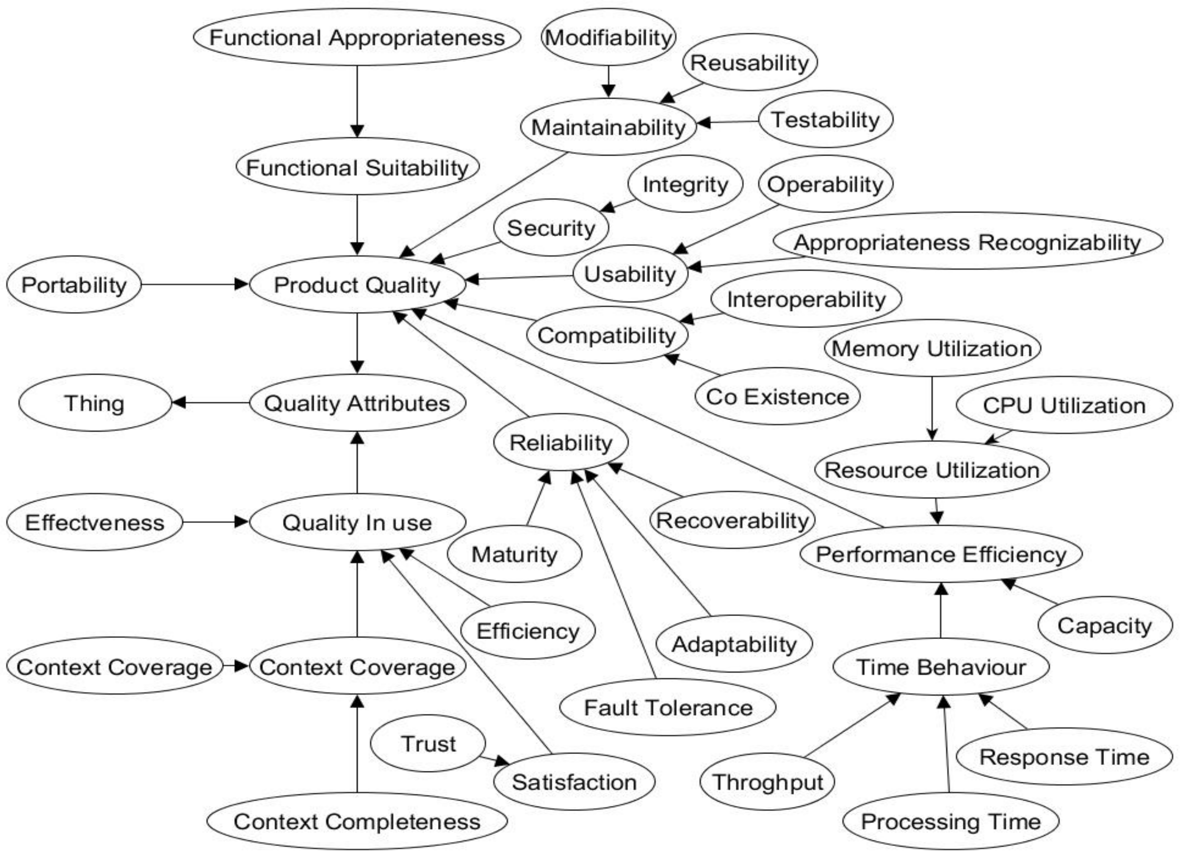


Figure 6-2 NFRs Taxonomy

An exception is 'Performance efficiency' which has two modified children 'Time behaviour' and 'resource utilization'. They are modified as compared with ISO/IEC 25010 standard's groupings to cover more quality characteristics of a software componenet. The modifications are as following:

- Resource utilization: it has been modified by adding two children, which are "Memory utilization" and "CPU utilization".

- Time behaviour: it has been modified by adding three children, which are "Response time", "Processing time" and "Throughput".

The added nodes are the only ones located at the fifth level of hierarchy. This level is also associated with an "is_a" relation to the forth level, which is indicated by the following Figure:

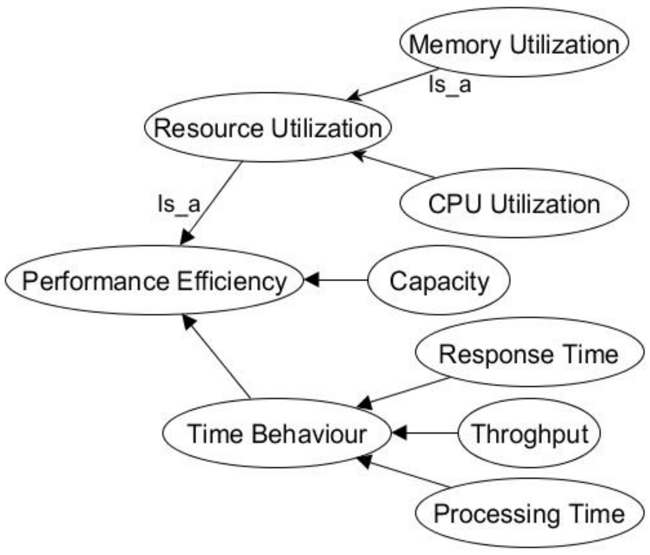


Figure 6-3 Performance Efficiency taxonomy

Moreover, there are more relations and information associated with nodes, which are captured by the ontological model i.e. NFRs relations and ranges. There are three types of relations, which support the algorithm query processing logic.

Let A and B represent NFR entities:

- A Has_a_Prerequisite B: B is required as a prior condition for A to happen or exist
- A Is_a_Prerequisite B: A is required as a prior condition for B to happen or exist
- A Is_a B: There is a true parent/child relationship between A and B and the child (or subclass) inherits directly from the parent (or superclass)

The following Figure indicates the relations associated with the NFR called Reliability, selected from the list of NFRs.

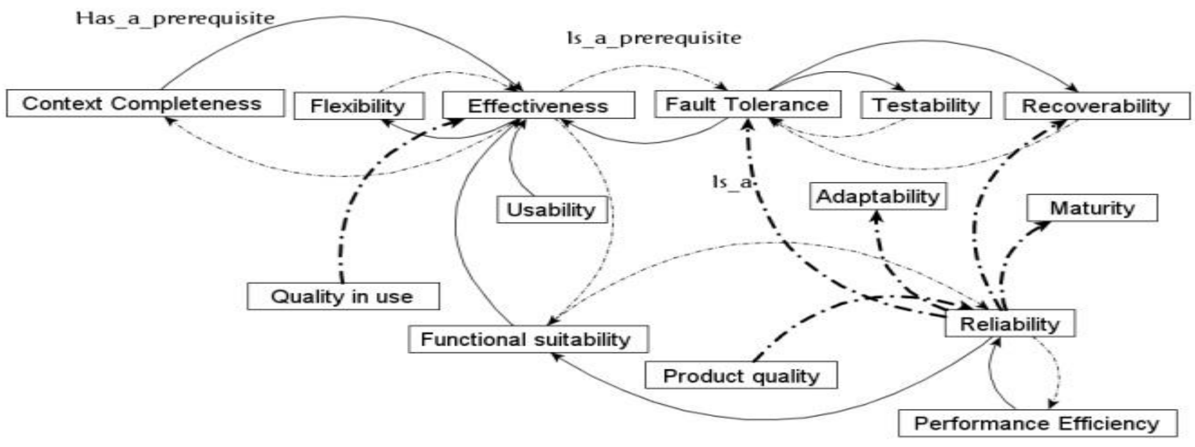


Figure 6-4 Relationship example of Reliability

The aim of Figure 6-4 is to demonstrate the complexity of NFRs. Three types of lines that indicate three types of relations among NFRs (mentioned above) are provided. The line that labels the "is_a" relation is based on ISO/IEC 25010 standard parent/child relationship [49]. The other two lines "has-a-prerequisite" and "is-a-prerequisite" show the prerequisites each NFR has. Prerequisites are elicited from the definition of NFRs. For example, the reliability definition (described in implementation Section 6.1.5.17) is dependent upon the following properties:

1. Functionality of component,
2. Condition of performance and
3. Time of performance.

According to the ISO/IEC 25010 standard, "Functional suitability" addresses the first requirement i.e. functionality of component and "Performance efficiency" addresses both condition and time of performance. As the Figure 6-4 is indicating, the NFRs' prerequisites have their own prerequisites and this is the main reason of complexity analysis (this has been addressed Chapter 4 in more details).

6.1.5 Non-Functional Requirements

The set of NFRs and their definitions are selected from the ISO/IEC 25010 standard's quality model [49]. This document includes quality characteristics related to software products. It helps to specify, measure and evaluate software product quality in general, but not specific, to software components. Thus, the general quality model is tailord to a component quality model. The ISO/IEC 25051 [46] standard that focuses specifically on component's quality requirements, but it only addresses test documentation, test cases and test reporting. In other words, the ISO/IEC 25051 standard aim, to certify that components perform as offered and delivered [122] but it does not offer any quality model.

In the following Section (6.1.5), a formula has been provided for each NFR (subject of the formula) to facilitates its range calculation. These formulas are proposed to calculate the the NFRs' range and they should not be used to measure NFRs. Morover, a formula has more than one variable. These variables are measured by a testing system (Section 8.2). The NFR measurement is out of scope of this study.

NFRs' ranges are calculated based on the information from ISO/IEC 25022 and 25023 [49, 51]. These standards do not specify any range for the NFRs. However, they provide an explanation of how to measure NFRs using provided measurement methods and formulas. The calculation method for each NFR's range is discussed below. Moreover, in order to improve ease of understanding, the ranges are specified in percentages rather than fractions.

6.1.5.1 Effectiveness

Effectiveness measures accuracy and completeness to which a goal can be achieved, when using a specific component. A component is completely effective when all its tasks are completed correctly. The effectiveness range is a closed interval of [0, 100]. The degree of effectiveness changes whenever some tasks are either partially complete or faile. To calculate the proportion of the tasks that are completed correctly, the number of tasks completed (C) is divided by the total number of tasks attempted (T). The calculation formula is indicated by the following equation:

$$Task\ Completion = \frac{C}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.2 Efficiency

Efficiency’s concerns are how the resources are consumed in relation to the accuracy and completeness to which a component can be used by users to achieve their intended outcomes. The efficiency range is a closed interval of [0, 100]. Time is the most common resource to complete a task. Component efficiency increases when the effectiveness increases and the task time decreases. To measure the efficiency, the proportions of the tasks that are completed correctly are divided by mean time spent. The calculation formula is indicated by the following equation:

$$Task\ Efficiency = \frac{Task\ Completion}{task\ time}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.3 Satisfaction / Trust / Usability

Satisfaction describes the degree to which a users’ needs are fulfilled when a product is used by them to achieve their intended outcomes. A component gains a high level of satisfaction when it is fully functional and useable. A psychometric questionnaire is used to assess user satisfaction. The format of a typical five-level psychometric questionnaire is a 5 point Likert scale: Strongly dissatisfied; dissatisfied; neither satisfied nor dissatisfied; satisfied; and strongly satisfied. Thus, the satisfaction range is a closed interval of [20, 100].

Trust is one of the satisfaction sub-characteristics. It indicates the level of assurance to a user that a component works as intended. The degree of trust depends on the result of provided test documentation, test cases and test reporting. To investigate whether users trust a component, a method similar to satisfaction (five-level psychometric questionnaire) is suggested. Thus, the trust range is same as the satisfaction range, which is a closed interval of [20, 100].

Usability is used to measure the degree to which a component is used by users to achieve listed outcomes with effectiveness, efficiency and satisfaction in a specific context of use. To investigate whether a component is usable, a method similar to satisfaction (five-level psychometric questionnaire) is suggested. Thus, the usability range is same as the satisfaction range, which is a closed interval of [20, 100].

6.1.5.4 Context Coverage / Context completeness

Context coverage is the degree to which a component is used with effectiveness, efficiency, freedom from risk, and satisfaction in both stated the context of use and in different contexts. Context completeness definition (as a sub-characteristic of context coverage) is similar to context coverage. However, its context is limited to specified context. Both context coverage and context completeness ranges is a closed interval of [0, 100]. In order to calculate the context of use proportion that a component is used, the number of contexts with unacceptable usability (N) is divided by a total number of contexts of use (T). The calculation formula is indicated by the following equation:

$$Context\ Measure = \frac{N}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.5 Flexibility

Flexibility is the degree to which a component is used with effectiveness, efficiency, freedom from risk, and satisfaction in contexts different to those initially specified in its requirements. Different contexts have different set of tasks, users and environment. Thus, a flexible component should be modifiable and adaptable to support different types of requests. The flexibility range is a closed interval of [0, 100]. In order to calculate the extent to which the component is used in the additional context of use, the number of additional contexts in which the component is usable (U) is divided by total number of additional contexts in which the component might be used (T). The calculation formula is indicated by the following equation:

$$Flexibility = \frac{U}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.6 Functional suitability

Functional suitability focuses on the ability of a component to meet stated and implied functionality when used under specified condition. The functional suitability range is a closed interval of [0, 100]. In order to calculate the proportion of the function that meets stated and implied needs, a formula based on similar quality attributes is designed. The calculation formula is indicated by the following equation, which presents SF as number of functions that meet stated and implied needs and T total number of functions:

$$Functional\ Suitability = \frac{SF}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.7 Functional appropriateness

Functional appropriateness focuses on the functionality of a component to facilitate the achievement of specified tasks and objectives. A component is functionally appropriate when a task completes only with the necessary steps, excluding any extra and unnecessary steps. The functional appropriateness range is a closed interval of [0, 100]. In order to calculate the number of implemented functions without any problem, the number of functions for pursuing specific tasks (NF) is divided by the number of functions from which a problem is detected (FP). The calculation formula is the following equation:

$$Functional\ Appropriateness = \frac{NF}{FP}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.8 Performance Efficiency

Performance efficiency is used to measure the performance relative to the amount of resources used under specified conditions. The performance efficiency range is a closed interval of [0, 100]. In order to calculate the performance efficiency of a component, a formula similar to the effectivness of ISO/IEC 25022 standard is designed (Section 6.1.5.1). The calculation includes two steps i.e. functional completeness calculation and mean time calculation. The number of functions completed (TF) is divided by the total number of functions attempted (MF). The calculation formula is indicated by the following equation:

$$Functional\ Completeness = \frac{TF}{MF}$$

The final calculation formula is indicated by the following equation:

$$Performance\ Efficiency = \frac{Functional\ Completeness}{time}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.9 Time Behaviour

Time behaviour facilitates measure the degree to which a component's set of response time, processing time and throughput of task performance meets the requirements. These three sub-characteristics are added to time behaviour in this study, which is not the case for the standard ISO model (Section 6.1.4). Thus, there is no equation to calculate their ranges.

- Processing time: the elapsed time in a system between receiving a request and sending the result.

- Response time: the time taken by the system to display results after a command entered. It includes processing time and transmission time.

The response and processing times are denoted as $t \in \mathbb{R}^+$ such that $t_m < t < t_n$ where t_m, $t_n \in \mathbb{R}^+$ and $t_m < t_n$ represents maximum and minimum response and processing times. Respectively, it is assumed that each process does not take more than 300000 milliseconds. Thus, these NFRs' range is a closed interval [1, 300000]. This range is only an assumption and does not indicate any fact. One millisecond was assumed to be the minimum of any time behaviour and 5 minutes (300000 millisecond) is a maximum for the time behaviour of a component.

- Throughput measures the number of activities a system can process in the specified time to achieve specific goals.

It is assumed that the range for throughput is a positive value for a signed binary integer, which is $2^{31} - 1$. Thus, the throughput range is a closed interval [0, 2147483647]. This range is defining a valid range for the throughput and it is only an assumption. The maximum range is selected as the max throughput being an int32. As a result the maximum valid range is 2147483647.

6.1.5.10 Resource utilization

Resource utilization facilitates measure the degree to which the amount of CPU and memory is used by a component when performing its tasks, or meets their requirements. In this study, two sub-characteristics, CPU utilization and memory utilization are added to resource utilization attribute of standard product model and they are not part of the standard model. Their range is indicated by a closed interval of [0, 100].

To calculate the CPU time used to perform a given task, ISO/IEC 25023 suggests dividing the operation time (OP) by the amount of CPU time actually used to perform a task (CPU). The calculation formula is indicated by the following equation:

$$CPU\ utilization = \frac{OP}{CPU}$$

In order to calculate the memory space used to perform a given task, the total amount of memory spaces (TM) is divided by the amount of memory spaces actually used to perform a task (SM). The calculation formula is indicated by the following equation:

$$Memory\ utilization = \frac{TM}{SM}$$

The evaluation of resource utilization attributes is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.11 Capacity

Capacity facilitates measure the degree to which the maximum limits of a component meets their requirements. It might be the capacity of item stored, the bandwidth, number of users and the size of database. The capacity range and its calculation method are same as the throughput (Section 6.1.5.9). It is assumed that the range for capacity is the positive value for a signed binary integer, which is $2^{31} - 1$. Thus, the capacity range is a closed interval of [0, 2147483647]. As described above, the range selected is an assumption. The maximum value of capacity is an int32, therefore, its valid maximum range is 2147483647

6.1.5.12 Compatibility

Compatibility is used to measure the degree to which a component can perform its duties while sharing the same environment. The compatibility range is a closed interval of [0, 100]. In order to check how compatible a component is in sharing its environment with others, the number of entities with which a component is compatible is divided by the number of entities that require compatibility. The compatibility is defined by the following equation:

$$Compatibility = \frac{\#\ of\ entities\ component\ is\ compatible\ with}{\#\ of\ entities\ require\ compatability}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.13 Co-existence

Co-existence is used to measure the degree to which a component performs its required functions efficiently, without negative impact on any other component while sharing a common environment and resources with other components. The Co-existence range is a closed interval of [0, 100]. To check how flexible a component is in sharing its environment with others, the number of entities with which component can co-exist as specified (CcoE) is divided by the number of entities that require co-existence (T). The definition of co-existance is provided by the following equation:

$$Co - existence = \frac{CcoE}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.14 Interoperability

Interoperability is used to measure the degree to which two or more components can exchange and use the information that has been exchanged. The interoperability range is a closed interval of [0, 100]. To calculate the interoperability of a component the following formula is provided:

$$Interoperability = \frac{\#\, of\ component\ with\ capability\ of\ having\ interaction}{\#\, of\ total\ components}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.15 Appropriateness recognisability

Appropriateness recognisability helps users to select a component that is fit for their intended use. In other words, appropriateness recognisability is a measure of a user comprehension of the capabilities of a component. The appropriateness recognisability range is a closed interval of [0, 100]. In order to calculate the proportion of functions that are described as understandable in the component description, the number of functions described as understandable in the component description (FD) is divided by a total number of functions (T). The appropriateness recognisability range is defined by the following equation:

$$Appropriateness\ recognisability = \frac{FD}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.16 Operability

Operability provides the ability to measure the degree to which a component has features that make it easy to function and control (properties of a component that make it work well in production). In order to calculate the degree of operability, the number of implemented functions which is customized during operation (CF) is divided by the number of functions requiring the customization capability (T). The calculation formula is indicated by the following equation:

$$Operability = \frac{CF}{T}$$

6.1.5.17 Reliability

Reliability focuses on the degree to which a component performs specified functions under stated conditions for a stated period of time. The reliability range is a closed interval of [0, 100]. The following formula defines the reliability of a component:

$$Reliability = \frac{\#\, of\ total\ defects}{\#\, of\ total\ functions}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.18 Maturity

Maturity is the capability of a component to meet its reliability needs under normal operation. In order to calculate the proportion of the number of corrected faults that have been detected, the number of corrected faults in design/coding/testing phase (NCF) is divided by the number of faults detected in a review or during testing (T). The maturity range is a closed interval of [0, 100]. Maturity is defined by the following equation:

$$Maturity = \frac{NCF}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.19 Fault tolerance

Fault tolerance indicates the capability of a component to maintain a specified performance level in cases of operation faults. The fault tolerance range is a closed interval of [0, 100]. In order to avoid critical and serious failures, the number of controlled fault patterns is calculated using the number of 'avoided critical and serious failure occurrence' against test cases of 'fault pattern (F)' divided by the number of executed test cases of fault pattern during testing (E). Fault tolerance is defined by the following equation:

$$Fault\ tolerance = \frac{F}{E}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.20 Recoverability

Recoverability measures the ability of a component to re-establish an acceptable level of performance and recover any or all affected data in the case of failure or interruption. The degree of effectiveness will change whenever some tasks are either partially completed or failed. The recoverability range is a closed interval of [0, 100]. The recoverability is defined by:

$$Recoverability = \frac{\#\ of\ recovered\ cases}{\#\ of\ total\ cases}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.21 Integrity

Integrity measures the level of protection from unauthorized access to data or modification of the program. The integrity range is a closed interval of [0, 100]. In order to calculate the level of protection against data corruption within a component, the number of data corruption instances that actually occurred (DC) is divided by the number of accesses where data corruption or data loss is expected to occur (X). Integrity is defined as:

$$Integrity = \frac{DC}{X}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.22 Reusability

Reusability measures the degree to which a component (asset) is used in more than one software/system. The reusability range is a closed interval of [0, 100]. The proportion of reusable assets is calculated by dividing the number of assets reused (RE) by the total number of assets in the reusable library (T) as defined by the following equation:

$$Reusability = \frac{RE}{T}$$

The reusability value is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.23 Modifiability

Modifiability focuses on the degree to which a component is effectively and efficiently modified without quality degradation or the introduction of faults. The modifiability range is a closed interval of [0, 100]. The modification success rate is calculated by dividing the number of problem within a certain period of time before maintenance (PB) by the number of problems in the same period after maintenance (PA). Modifiability is defined as:

$$Modifiability = \frac{PB}{PA}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.24 Testability

Testability measures the effectiveness and efficiency with which test criteria can be created for a component and tests is executed to test the achievability of those criteria. The testability range is a closed interval of [0, 100]. In order to investigate the testability of a component, the number of test functions implemented as a specification (C) is divided by number of required test functions (F). Testability is defined as:

$$Testability = \frac{C}{F}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.25 Maintainability

Maintainability measures the degree of effectiveness and efficiency with which a component is modified. It defines the level of effort required to modify a component. The maintainability range is a closed interval of [0, 100]. Maintainability is defined as:

$$Maintainability = \frac{\#\ of\ fuctions\ affected\ by\ a\ change}{\#\ of\ total\ functions}$$

6.1.5.26 Portability

Portability measures the degree of effectiveness and efficiency with which a component is relocated from one software system, hardware assembly, or environment to another. The portability range is a closed interval of [0, 100]. Portability is defined by the following formula:

$$Portability = \frac{\#\ of\ working\ relocated\ functions}{\#\ of\ total\ relocated\ fucntions}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

6.1.5.27 *Adaptability*

Adaptability measures the degree of effectiveness and efficiency with which a component is adapted from one software, hardware or another environment to another. The adaptability range is a closed interval of [0, 100]. In order to investigate the adaptability of a component during combined operating testing, the number of operational functions when tasks are not completed or are not enough resulted to meet the adequate level (F) is divided by total number of functions which were tested (T). Adaptability is defined by the following equation:

$$Adaptability = \frac{F}{T}$$

The solution is a closed interval [0, 1], a real number that is greater than or equal to 0 and less than or equal to 1.

The non-functional requirement concepts predicted in the ontology and their ranges is listed in the Table 6-1:

Name	Range
Effectiveness	[0, 100]
Efficiency	[0, 100]
Satisfaction or Trust	[20 , 100]
Context Coverage or Context completeness	[0, 100]
Flexibility	[0, 100]
Functional suitability	[0, 100]
Functional appropriateness	[0, 100]
Performance Efficiency	[0,100]
Response time	[1 mls, 300000]
Processing time	[1 mls, 300000]

Name	Range
Throughput	[0, 2147483647]
CPU utilization	[0, 100]
Memory utilization	[0,100]
Capacity	[0, 2147483647]
Compatibility or Co-existence	[0,100]
Interoperability	[0,100]
Usability	[20 , 100]
Appropriateness recognisability	[0, 100]
Operability	[0, 100]
Reliability	[0, 100]
Maturity	[0, 100]
Fault tolerance	[0, 100]
Recoverability	[0, 100]
Integrity in	[0, 100]
Reusability	[0, 100]
Modifiability	[0, 100]
Maintainability or testability	[0, 100]
Portability	[0, 100]
Adaptability	[0, 100]

Table 6-1 Non-functional Requirement Ranges

6.2 Implementation

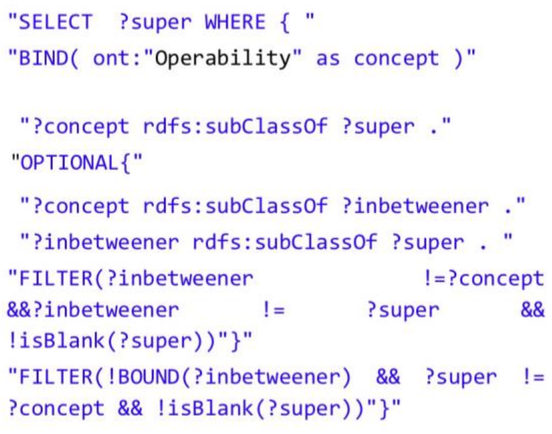
6.2.1 Tools and Techniques

The main programming language for implementing this study's prototype is Java. This study employs different tool and techniques for automated identification and selection of components.

Algorithm: It is designed to calculate the score of each component description as a comparison factor and then select the component with the higher score. The calculation rules (logic) of algorithm explained in "Calculate Score" use case (Section 6.3.3).

Jena: it is a Java framework [123] for building Semantic Web applications. It provides a programmatic environment for RDF [118], OWL [91], SPARQL [120] and includes reasoner and inference engine. The Jena Framework includes: an RDF API, in-memory and persistent storage and a SPARQL query engine

The RDF API provides the required interface for reading a RDF file. This API is used as RDF parser and serializer. The following is a sample SPARQL query that finds the parent node of the "operability" node (a non-functional requirement) in an RDF graph.

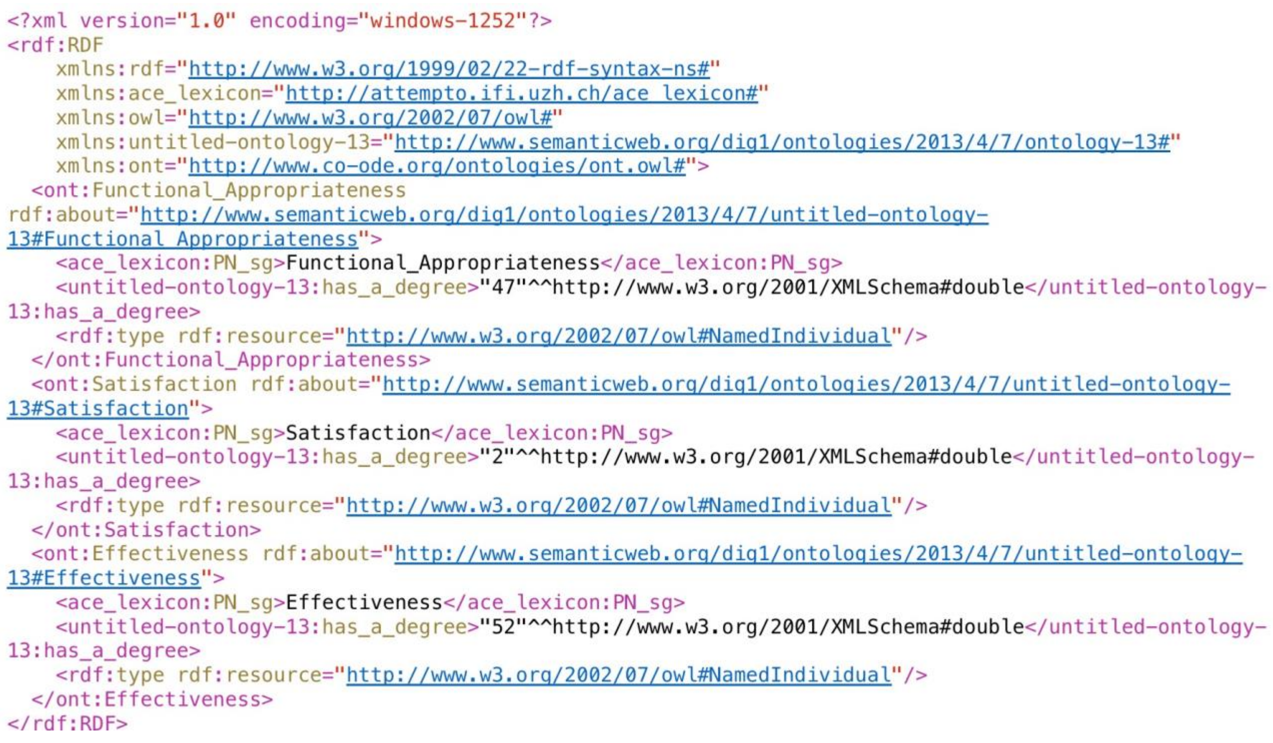


```
"SELECT  ?super WHERE { "
"BIND( ont:"Operability" as concept )"

 "?concept rdfs:subClassOf ?super ."
"OPTIONAL{"
 "?concept rdfs:subClassOf ?inbetweener ."
 "?inbetweener rdfs:subClassOf ?super . "
"FILTER(?inbetweener              !=?concept
&&?inbetweener       !=       ?super       &&
!isBlank(?super))"}"
"FILTER(!BOUND(?inbetweener)  &&  ?super  !=
?concept && !isBlank(?super))"}"
```

Figure 6-5 Find Parent Node Query

The RDF API also provides the required interface for writing a RDF file. A component description specifies the non-functional capabilities of a component. It contains a set of non-functional names paired with values. Each name represents the actual quality of the component and the value indicates the degree desired for that component quality that has defined range as described in the Section 6.5.1. The degree is specified in percentage format. The following is a sample RDF file (Component Description):



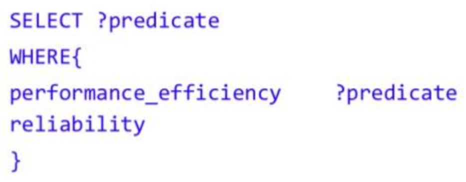
```
<?xml version="1.0" encoding="windows-1252"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ace_lexicon="http://attempto.ifi.uzh.ch/ace_lexicon#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:untitled-ontology-13="http://www.semanticweb.org/dig1/ontologies/2013/4/7/ontology-13#"
    xmlns:ont="http://www.co-ode.org/ontologies/ont.owl#">
  <ont:Functional_Appropriateness
rdf:about="http://www.semanticweb.org/dig1/ontologies/2013/4/7/untitled-ontology-
13#Functional_Appropriateness">
    <ace_lexicon:PN_sg>Functional_Appropriateness</ace_lexicon:PN_sg>
    <untitled-ontology-13:has_a_degree>"47"^^http://www.w3.org/2001/XMLSchema#double</untitled-ontology-
13:has_a_degree>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  </ont:Functional_Appropriateness>
  <ont:Satisfaction rdf:about="http://www.semanticweb.org/dig1/ontologies/2013/4/7/untitled-ontology-
13#Satisfaction">
    <ace_lexicon:PN_sg>Satisfaction</ace_lexicon:PN_sg>
    <untitled-ontology-13:has_a_degree>"2"^^http://www.w3.org/2001/XMLSchema#double</untitled-ontology-
13:has_a_degree>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  </ont:Satisfaction>
  <ont:Effectiveness rdf:about="http://www.semanticweb.org/dig1/ontologies/2013/4/7/untitled-ontology-
13#Effectiveness">
    <ace_lexicon:PN_sg>Effectiveness</ace_lexicon:PN_sg>
    <untitled-ontology-13:has_a_degree>"52"^^http://www.w3.org/2001/XMLSchema#double</untitled-ontology-
13:has_a_degree>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
  </ont:Effectiveness>
</rdf:RDF>
```

Figure 6-6 Sample RDF File

Conceptual model (composed of ontological and data models)**:** the Web Ontology Language (OWL) Full [91] is used for authoring the NFRs knowledge base, it is compatible with RDF schema. Apache Jena is used to extract data from, or write to, the model. The RDF files represent the data (instances), which are in the form of RDF statements (subject-predicate-object). Subject is what the statement is about (in our case NFR name), predicate is the property, and object is the value of the statement.

Automated reasoning: The internal reasoner of the Jena framework supports the process of deriving additional information through term inference [123]. It represents relationships that are implicit in the ontology based on explicitly stated relationships and implement semantics internally. Reasoners provide a means of making inferences based on facts depicted as RDF statements. This allows the algorithm (query processing logic) to infer new facts.

Semantic query language: The language that facilitates query functionality over the RDF and OWL models is SPARQL [120]. It employs ARQ [124] which is a Java-based framework for executing queries against other two tiers (ontology and data models). SPARQL query makes use of variables and conditions. In the following simple query example the variables are associated with RDF terms (NFR names)



```
SELECT ?predicate
WHERE{
performance_efficiency    ?predicate
reliability
}
```

Figure 6-7 Find Realtionship Between Two Nodes

This query on our data has the following result: is_a_prerequisite, which is a relationship between two NFRs (efficiency and reliability).

6.2.2 Object-Oriented Analysis

In this Section, a different level of abstraction is illustrated by the UML [125] class diagram to document the implementation of the prototype. The highest level is the context level of application which it models the high-level functionality of the system and also its interaction with the outside world. Moreover, the interactions within a class and or its associated objects are modeled.

To implement the system, Unified Modelling Language (UML) and SDLC [126] are used. Furthermore, the abstractive definitions of the most important objects are used for the object-oriented modelling of the system. The building blocks of the object-orientated system are:

- Component description class: responsible for keeping and managing each component's properties such as weight, name and semantic status. Moreover, it generates the methods such as score calculation and weight calculation.
- Non-functional requirement class: responsible for keeping and managing each NFR properties and find the NFRs dependencies such as: finding the parent or child of NFRs selected by the user.
- Prerequisites class: responsible for finding each NFR prerequisites. In order to do this, it needs to have full access to the non-functional requirement class as its subclass.
- Query class: performs query reformulation on user query. Moreover, it communicates with all parts of the component selection system to assist selection of the best result for the user.
- Euclidean distance class: responsible for calculating the distance between system results to the user requested NFRs. In order to perform mathematical operations such as the Euclidean distance calculation in multidimensional space, the Apache Commons mathematics library [127] is used. It is an open source optimized library.
- Data transfer object (DTO) classes: There are two DTO classes, which are responsible for transferring data between different processes. These classes are called a) Sort Function Class, which helps sort the components based on their score, and b) Two Return Value Class, which helps the time needed to return more than one item with the method call. Figure 6-8 shows all objects of the designed system:

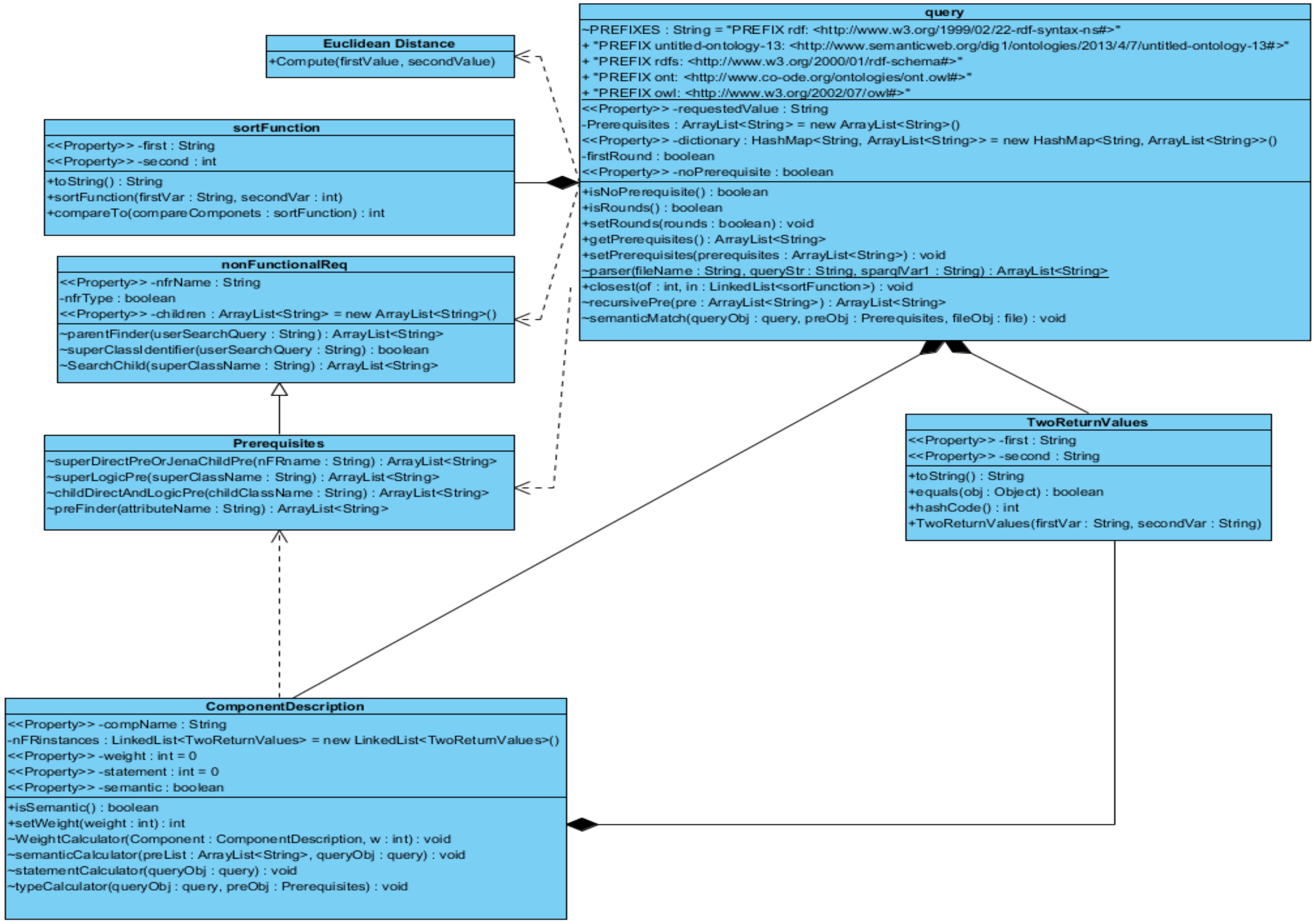

Euclidean Distance
+Compute(firstValue, secondValue)
sortFunction
<<Property>> -first : String
<<Property>> -second : int
+toString() : String
+sortFunction(firstVar : String, secondVar : int)
+compareTo(compareComponets : sortFunction) : int
nonFunctionalReq
<<Property>> -nfrName : String
-nfrType : boolean
<<Property>> -children : ArrayList<String> = new ArrayList<String>()
~parentFinder(userSearchQuery : String) : ArrayList<String>
~superClassIdentifier(userSearchQuery : String) : boolean
~SearchChild(superClassName : String) : ArrayList<String>
Prerequisites
~superDirectPreOrJenaChildPre(nFRname : String) : ArrayList<String>
~superLogicPre(superClassName : String) : ArrayList<String>
~childDirectAndLogicPre(childClassName : String) : ArrayList<String>
~preFinder(attributeName : String) : ArrayList<String>
query
~PREFIXES : String = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
+ "PREFIX untitled-ontology-13: <http://www.semanticweb.org/dig1/ontologies/2013/4/7/untitled-ontology-13#>"
+ "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"
+ "PREFIX ont: <http://www.co-ode.org/ontologies/ont.owl#>"
+ "PREFIX owl: <http://www.w3.org/2002/07/owl#>"
<<Property>> -requestedValue : String
-Prerequisites : ArrayList<String> = new ArrayList<String>()
<<Property>> -dictionary : HashMap<String, ArrayList<String>> = new HashMap<String, ArrayList<String>>()
-firstRound : boolean
<<Property>> -noPrerequisite : boolean
+isNoPrerequisite() : boolean
+isRounds() : boolean
+setRounds(rounds : boolean) : void
+getPrerequisites() : ArrayList<String>
+setPrerequisites(prerequisites : ArrayList<String>) : void
~parser(fileName : String, queryStr : String, sparqlVar1 : String) : ArrayList<String>
+closest(of : int, in : LinkedList<sortFunction>) : void
~recursivePre(pre : ArrayList<String>) : ArrayList<String>
~semanticMatch(queryObj : query, preObj : Prerequisites, fileObj : file) : void
TwoReturnValues
<<Property>> -first : String
<<Property>> -second : String
+toString() : String
+equals(obj : Object) : boolean
+hashCode() : int
+TwoReturnValues(firstVar : String, secondVar : String)
ComponentDescription
<<Property>> -compName : String
-nFRinstances : LinkedList<TwoReturnValues> = new LinkedList<TwoReturnValues>()
<<Property>> -weight : int = 0
<<Property>> -statement : int = 0
<<Property>> -semantic : boolean
+isSemantic() : boolean
+setWeight(weight : int) : int
~WeightCalculator(Component : ComponentDescription, w : int) : void
~semanticCalculator(preList : ArrayList<String>, queryObj : query) : void
~statementCalculator(queryObj : query) : void
~typeCalculator(queryObj : query, preObj : Prerequisites) : void

Figure 6-8 Components Selection System Class Diagram

6.3 Use-Cases

The core requirements for this study's prototype, actors and system components, are modelled using the use-case diagrams. Next, the primary functionality of the system is described in the sequence diagrams. A use-case diagram provides a high-level idea of what functionality the system provides. Figure 6-9 shows the component selection system's use case diagram.

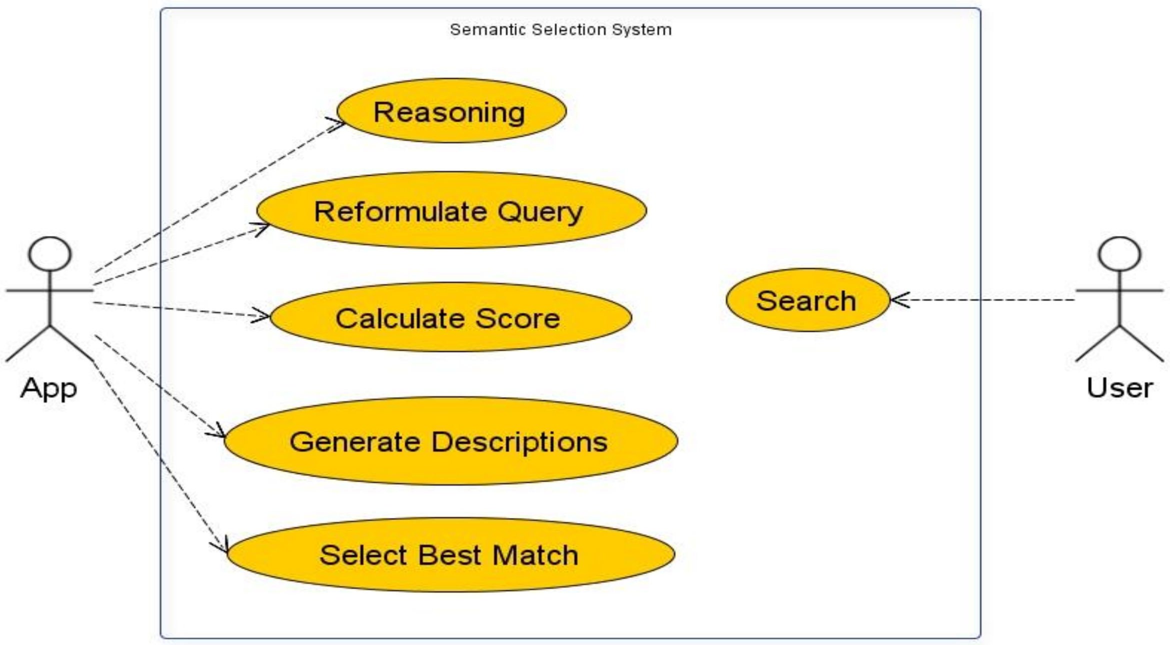


Figure 6-9 Semantic Selection System Use Case Diagram

6.3.1 Reasoning Use Case

In order to automate the analysis of non-functional requirements, the reasoning use case captures and provides the required knowledge. It's methods implement the reasoning functionality. This method employs the Apache Jena framework for model management tasks (read, write and integration) and reasoning purposes. It creates in-memory models (graphs) for ontology (OWL), data (RDF component descriptions) and the inferred model. The Jena reasoner requires the data (RDF models) and user's query to generate the inferred model and derive knowledge. Therefore, this method also converts the user's query to an executable query format (SPARQL). The Table 6-2 and Figure 6-10 illustrate information about and execution of this use case:

Use Case	**Deatails**
Requirement	Fulfilling the decision support technique for component selection
Goal in context	Generate relations from available knowledge
Successful end condition	Relations are ready to use
Fail end condition	Scores cannot be calculated
Actor	Application
Pre-condition	The user should design query
Trigger	The user clicks submit button

Table 6-2 Reasoning Use Case

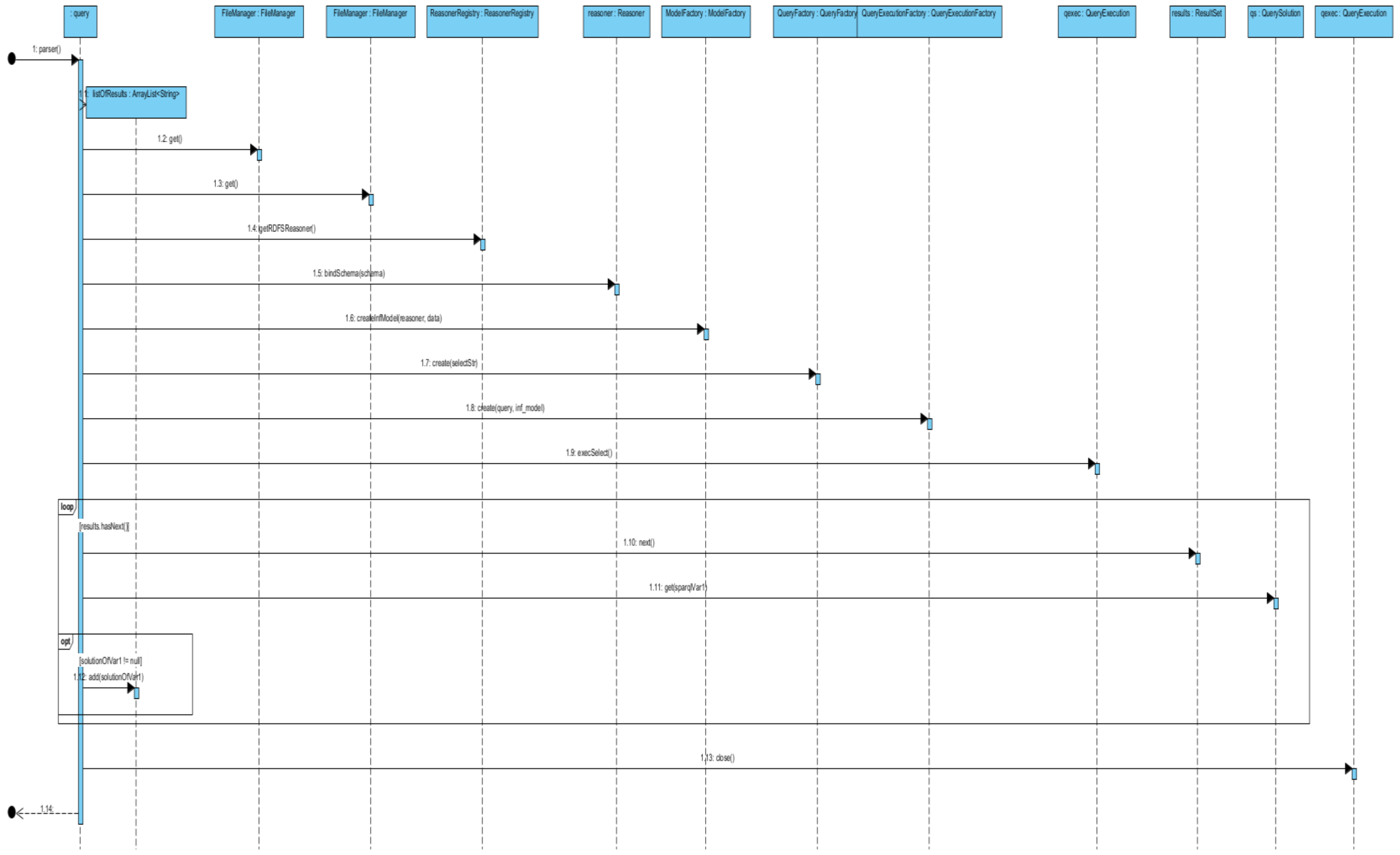


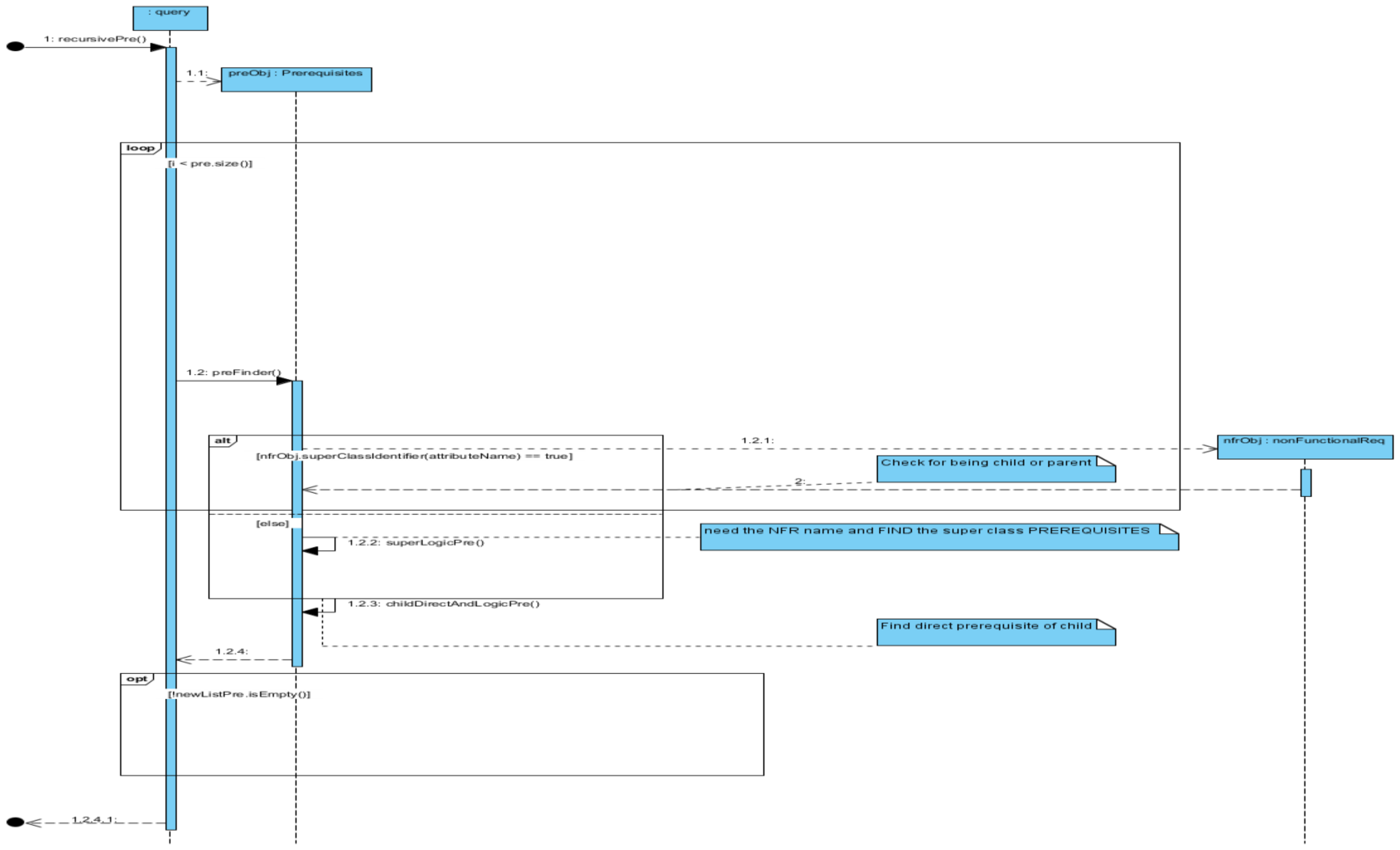
Figure 6-10 Sequence Diagram of Reasoning

6.3.2 Reformulate Query Use Case

In order to select the best component, the system needs to check the quality of user’s query and improve it semantically. This use case finds and adds the missing prerequisites of NFRs to the user’s query. Firstly; it distinguishes the type of NFR (as discussed in 6.1.4, there are two types of NFRs: parent and child). Secondly, if the NFR is a parent class then its prerequisites are direct prerequisites of both parent class and its children. However, if the NFR is a child class then its prerequisites are just direct prerequisites of the child class. The Table 6-3 and Figure 6-11 illustrate information about and execution of this use case:

Use Case	**Deatails**
Requirement	Query improvement
Goal in context	Add prerequisites of query
Successful end condition	Query is ready to send to application
Fail end condition	Prerequisite inspection fail and finally the logic fail
Actor	Application
Pre-condition	Relations should be analyzed
Trigger	It triggers when there is an element (NFR) with some prerequisites

Table 6-3 Reformulate Query Use Case



: query
1: recursivePre()
1.1:
preObj : Prerequisites
loop
[i < pre.size()]
1.2: preFinder()
alt
[nfrObj.superClassIdentifier(attributeName) == true]
1.2.1:
nfrObj : nonFunctionalReq
Check for being child or parent
2:
[else]
1.2.2: superLogicPre()
need the NFR name and FIND the super class PREREQUISITES
1.2.3: childDirectAndLogicPre()
Find direct prerequisite of child
1.2.4:
opt
[!newListPre.isEmpty()]
1.2.4.1:

Figure 6-11 Sequence Diagram of Reformulate Query

6.3.3 Calculate Score Use Case

In order to retrieve a semantically valid component, there is a mechanism to validate them against the rules and score them accordingly. This use case describe the components indexing. The following examples explain different situations in which the calculation rules assist ranking the component descriptions.

Rule 1: The component description that contains the highest number of prerequisites will earn a higher score. This score is then added to other scores (at the end of the process the component description with highest score is selected).

There is an "is_a" relationship among the classes, which makes all child instances equal to their parent instance. If Class and Class1 are classes, then Class is_a Class1 means that every instance of Class, at any time, is an instance of Class1 at the same time. Therefore, the Appropriateness Recognizably and Operability attributes both have an "is-a" relationship with the Usability attribute then Appropriateness Recognizably individual (instance) and Operability individuals are treated exactly as a usability individual. In this case, the system takes care of the prerequisites of the child classes as well, if the requested NFR is a super class (Usability).

Rule 2: If the requested NFR is a super class then the component description that contains all of its child prerequisites will earn a greater score.

In some cases, for NFRs such as Reliability, the component description must contain chains of related NFR s (toward their prerequisites). This situation occurs when the requested NFR has a series of relations. First NFR has a prerequisite and it is a super class. Second NFR is itself a super class with some prerequisites as well, therefore, it is circular. Circular referencing includes a series of references where the last object references the first, resulting in a closed loop.

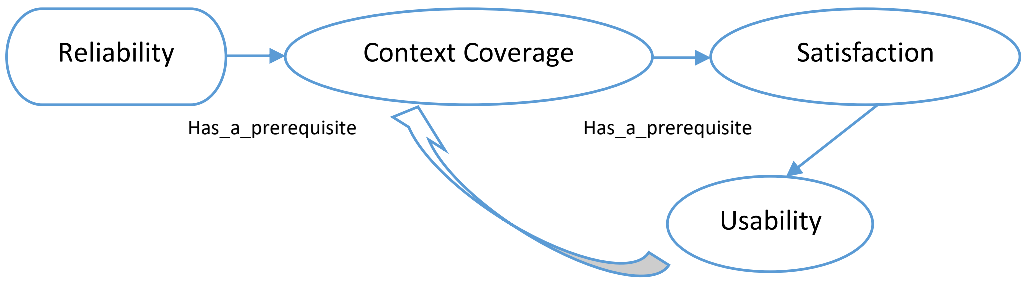


Figure 6-12 Circular Referencing

Rule 3: A component description is considered "matched semantically": if a complete chain of elements exists in the component description (the ideal case).

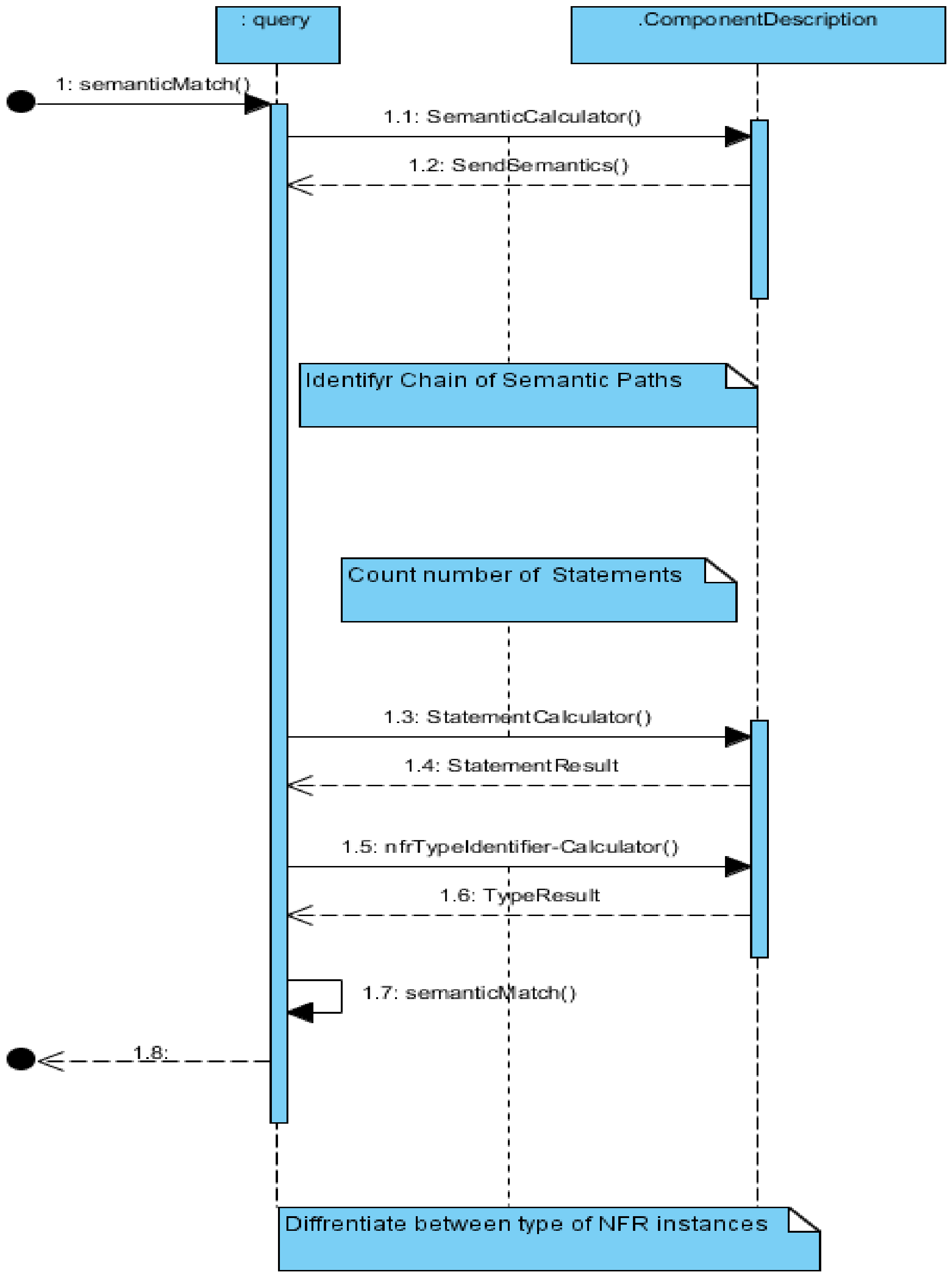
Rule 4: The component description with more statements (Subject, Predicate and Object) will earn a greater score.

There are two types of prerequisites: direct, which is the prerequisite of the class itself and indirect, which is a prerequisite of its subclass.

Rule 5: A direct prerequisite has a better weight in comparison with an indirect one. The sum of these weights will determine the acceptable component description. The Table 6-4 and Figure 6-13 illustrate information about, and execution of, this use case:

Use Case	**Deatails**
Requirement	Component identification
Goal in context	Component indexing
Successful end condition	Component indexing will be finished
Fail end condition	The query for the best match rejected
Actor	Application
Pre-condition	Prerequisites should get found and semantic statements should get counted. In order to have this condition, relationship should discover
Trigger	Relationships part completed

Table 6-4 Calculate Score Use Case



: query
.ComponentDescription
1: semanticMatch()
1.1: SemanticCalculator()
1.2: SendSemantics()
Identifyr Chain of Semantic Paths
Count number of Statements
1.3: StatementCalculator()
1.4: StatementResult
1.5: nfrTypeIdentifier-Calculator()
1.6: TypeResult
1.7: semanticMatch()
1.8:
Diffrentiate between type of NFR instances

Figure 6-13 Sequence Diagram of Calculate Score

6.3.4 Generate Description Use Case

A component repository is built to test the approach advocated. The repository is populated with RDF files. The Generate Description use case creates RDF files that act as the component descriptions. The component descriptions provide the conceptual instance of the ontological model. An RDF document can contain more than one statement (subject, predicate and object). Every component description contains a set of NFRs, which are instantiated from the NFRs concepts predicted in the ontology and their ranges. Ranges are listed in the Table 6-1, which acts as an important collection of data for developing the rest of the system. The Table 6-5 and Figure 6-14 illustrate information about and execution of this use case:

Use Case	**Deatails**
Requirement	Data collection
Goal in context	Provide the conceptual instance of ontological model
Successful end condition	We can have a repository full of data
Fail end condition	No data
Actor	Application
Pre-condition	Define NFRs rages
Trigger	Run as Java application

Table 6-5 Generate Description Use Case

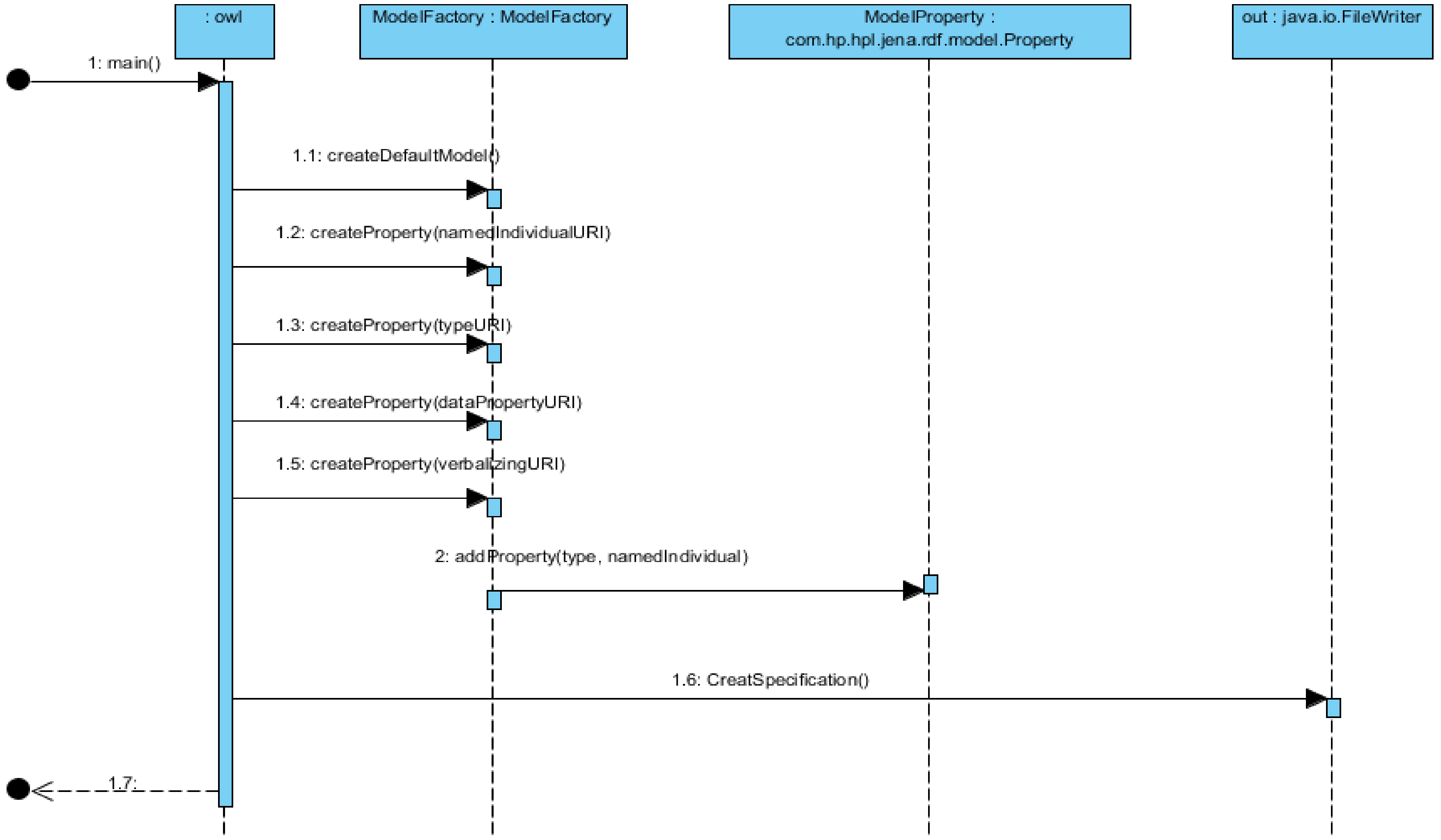

: owl
ModelFactory : ModelFactory
ModelProperty : com.hp.hpl.jena.rdf.model.Property
out : java.io.FileWriter
1: main()
1.1: createDefaultModel()
1.2: createProperty(namedIndividualURI)
1.3: createProperty(typeURI)
1.4: createProperty(dataPropertyURI)
1.5: createProperty(verbalizingURI)
2: addProperty(type, namedIndividual)
1.6: CreatSpecification()
1.7:

Figure 6-14 Sequence diagram of Generate Description

6.3.5 Select Best Match Use Case

The aim of the Select Best Match use case is to match a component with user query. There are some cases where the system may return multiple matches (components with the same highest score). The algorithm chooses the closest one (among the components with highest score) to the user query. The closest component is the one whose NFRs have the minimum Euclidean distance to the user values in comparison to other matches. In mathematics, the Euclidean distance is a straight-line distance between two points in Euclidean space. Using the Euclidean distance method, this use case calculates the distance between user point (vector) and the second point in Euclidean space (NFR point). User point or vector is a value that is extracted from the user query for the selected NFR (by user) and NFR point is the defined value of each NFR in the designed program. The distance between these two points is the length of the path connecting them. The Table 6-6 and Figure 5-15 illustrate information about and execution of this use case.

Use Case	**Deatails**
Requirement	If there were more than a result, the system should be able to choose the most qualified one
Goal in context	To choose from components with the same score
Successful end condition	Show the best match
Fail end condition	Show group of components with same highest score
Actor	Application
Pre-condition	Component’s result should be sorted
Trigger	If result list has more than one member

Table 6-6 Select best match Use Case

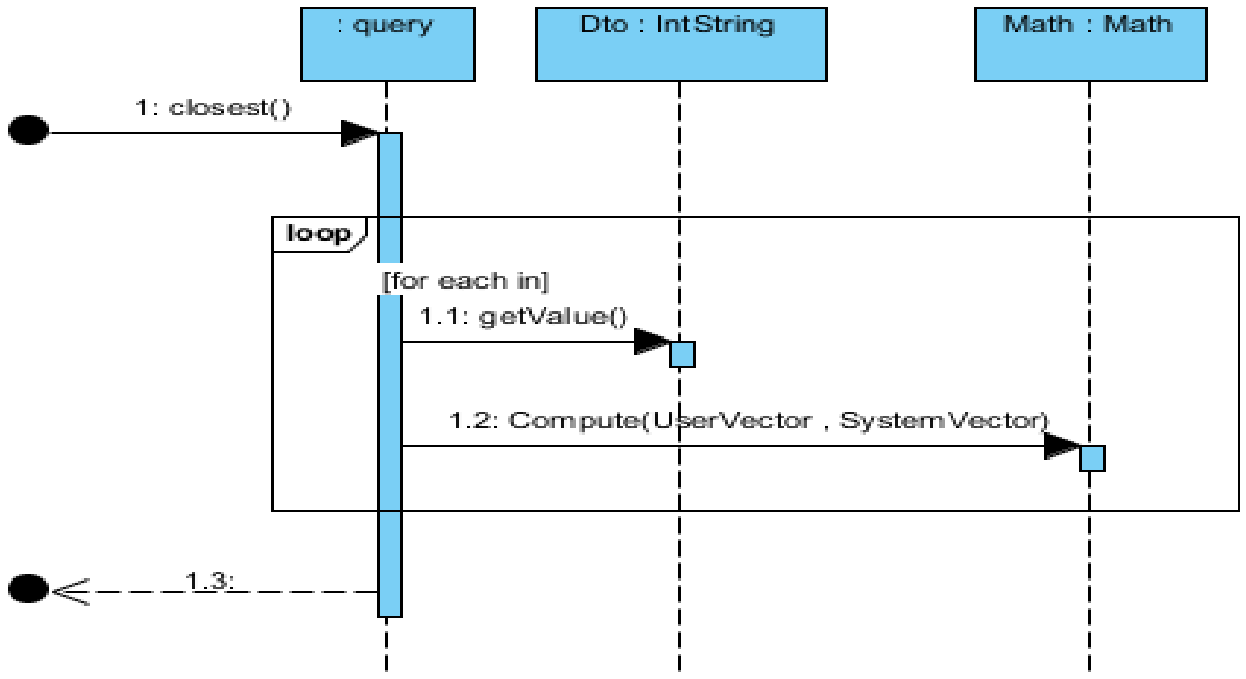


Figure 6-15 Sequence Diagram of Select Best Match

6.3.6 Search Use Case

The functionality implemented for the search use case is used as the core infrastructure for information access and reporting in the prototype. This use case is responsible for collecting user inputs and sending them to the use cases described above (sections 6.3.1-6.3.5). The Table 6-7 and Figure 6-16 illustrate information about and execution of this use case.

Use Case	**Deatails**
Requirement	Finding the proper software component
Goal in context	See the system output
Successful end condition	System will be able to produce some result
Fail end condition	No result
Actor	User
Pre-condition	User should provide pair of NFR's name and value
Trigger	No trigger

Table 6-7 Search Use Case

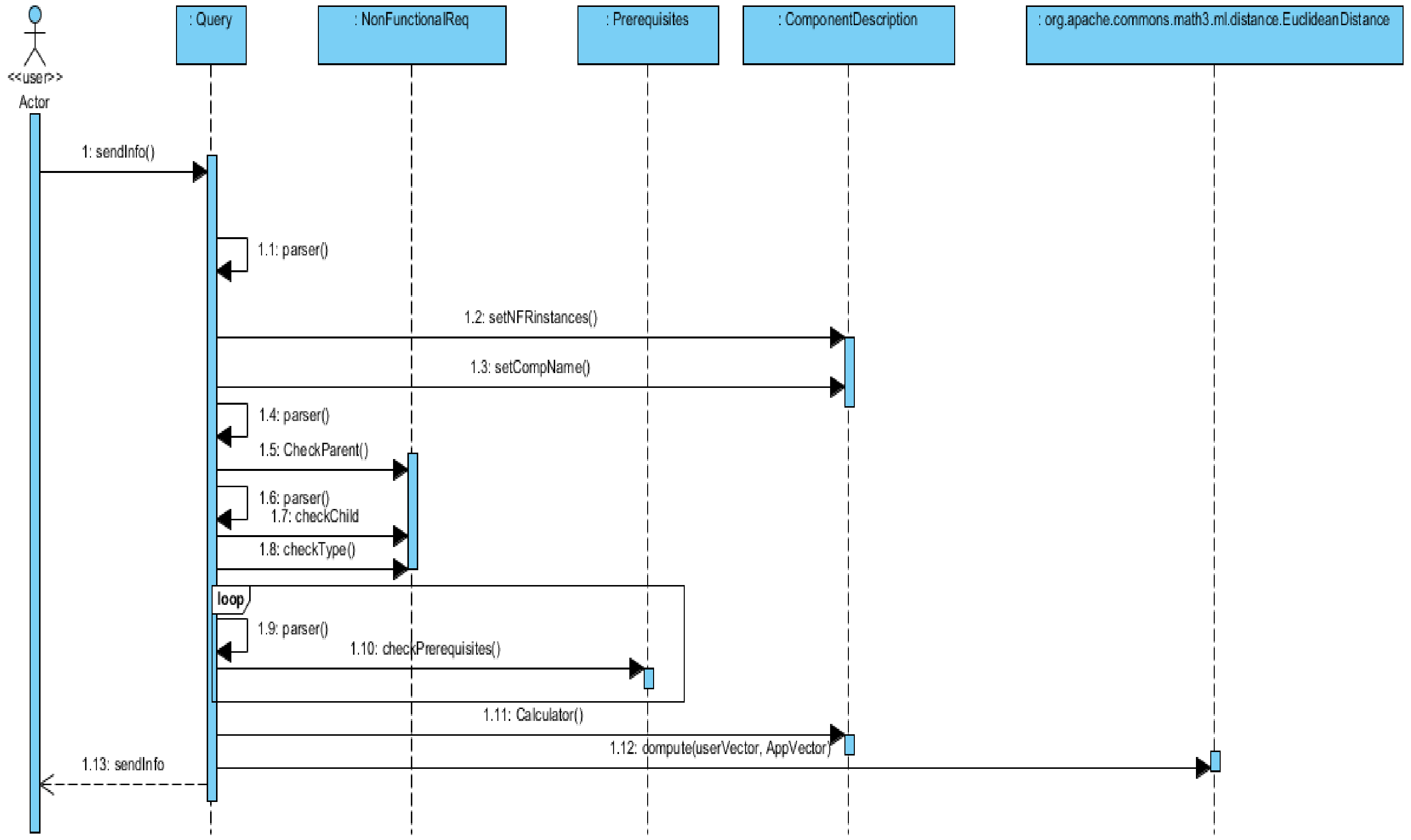


Figure 6-16 Sequence Diagram of Search

This chapter summarized the implementation and design of our prototype components. UML was used to model the important objects. In addition, different parts of the architecture are described. In the next chapter, an experimental case study is presented to validate the approach.

7 Discussion

As discussed in previous chapters, CBSD is attractive as a software development approach because it reduces the cost and time of development and increase the quality of the produced software system [21-24]. However, some issues still remain, in particular when focusing on component retrieval, where there is a need for a component retrieval process based on both functional and non-functional requirements.

Requirement satisfaction is difficult due to component selection being an iterative process. The result of component selection depends on the success of its classification and retrieval mechanism, where a wide variety of component repositories are considered for performing component search. The component either might not be found or, when found, might not perform the specific function, or fail to interoperate with other components [21].

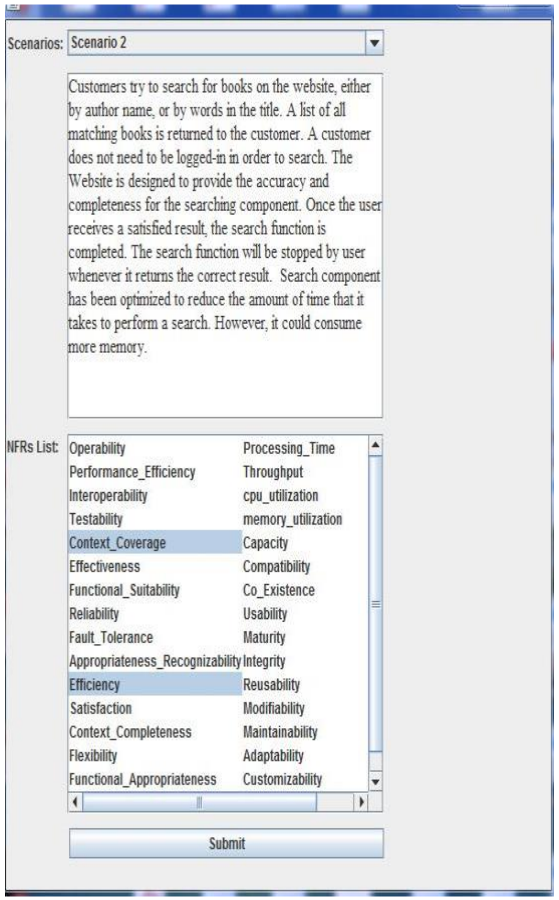
In this thesis, a framework to address these issues is presented. The framework helps to select and identify components semantically. A non-functional requirement ontology has been employed as a conceptual model for reasoning about component descriptions, and a search algorithm that matches the best component according to the reasoning process outputs has been implemented. A research question in this thesis was whether the use of a non-functional requirement ontology can support accurate component identification. Pursuing this further, repositories in different sizes are created. They are populated by designing a program that generates these component descriptions in quantities of 50, 100, 200, 400, 600, 800 and 1000 components. The component descriptions are generated in the form of RDF files due to its machine readability. A mix of qualitative and quantitative analysis has been selected for the data. Qualitative analysis helps to explore new knowledge and theories, while quantitative analysis helps to test the method’s performance.

A prototype tool for searching and retrieving from an example component repository is built to verify the approach. In this tool, a non-functional requirement ontology is provided and translated into OWL. The tool is implemented in Java and the experiments are run on Windows 7 (system specification is discussed in Section 7.2). The tool has a simple interface (Figures 7.1-7.3), where the user could choose the query element from the non-functional requirements list, and fill in the non-functional requirements' values into the text fields based on the defined ranges in the Section 6.1.5. In the back-end, the algorithm is responsible for performing the following tasks whenever required:

- Searching the OWL document for the user's query relationship analysis, and
- Connecting to the repository for the component descriptions inspection

The user query is classified using different ontology properties. The relationship among ontology concepts is added semantic information to the query. The algorithm analyses each non-functional requirement in terms of its relations (e.g. prerequisites) and types (e.g. general or specific) during the initial step. Then, according to the analysis result, it adds the required prerequisites to the query in order to perform a semantic search. Component descriptions are inspected in terms of non-functional requirements and their prerequisites. Each component description may have 1 to 32 prerequisites (non-functional requirements). Consequently, a component not matching any of the user's query elements and their prerequisites is eliminated.

Finally, the result is presented to the user in the form of non-functional requirement name-value pairs. The name of each non-functional requirement consists of the user's requested non-functional requirement and their available prerequisites. Each non-functional requirement value indicates the degree of non-functional requirements. Sometimes more than one result is possible but the algorithm selects the one closest to the user request by the help of Euclidean Distance formula (Section 6.3.5).



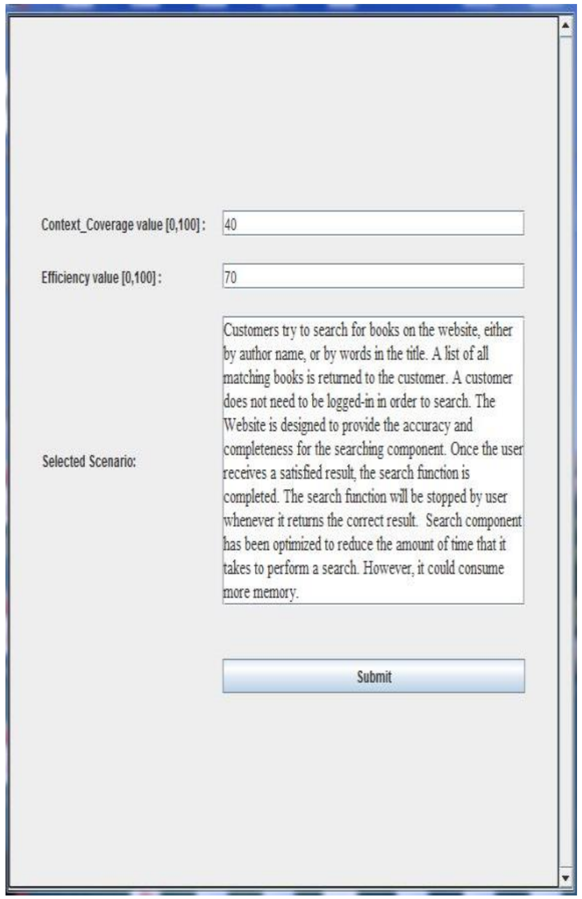
Scenarios: Scenario 2

Customers try to search for books on the website, either by author name, or by words in the title. A list of all matching books is returned to the customer. A customer does not need to be logged-in in order to search. The Website is designed to provide the accuracy and completeness for the searching component. Once the user receives a satisfied result, the search function is completed. The search function will be stopped by user whenever it returns the correct result. Search component has been optimized to reduce the amount of time that it takes to perform a search. However, it could consume more memory.

NFRs List:

Operability	Processing_Time
Performance_Efficiency	Throughput
Interoperability	cpu_utilization
Testability	memory_utilization
Context_Coverage	Capacity
Effectiveness	Compatibility
Functional_Suitability	Co_Existence
Reliability	Usability
Fault_Tolerance	Maturity
Appropriateness_Recognizability	Integrity
Efficiency	Reusability
Satisfaction	Modifiability
Context_Completeness	Maintainability
Flexibility	Adaptability
Functional_Appropriateness	Customizability

Figure 7-1 Initial Panel

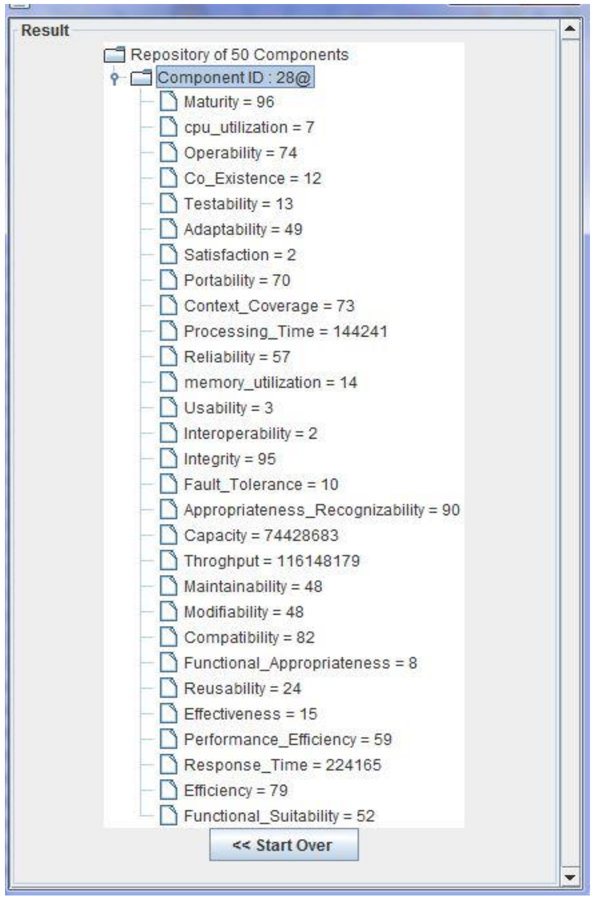


Context_Coverage value [0,100] : 40

Efficiency value [0,100] : 70

Selected Scenario: Customers try to search for books on the website, either by author name, or by words in the title. A list of all matching books is returned to the customer. A customer does not need to be logged-in in order to search. The Website is designed to provide the accuracy and completeness for the searching component. Once the user receives a satisfied result, the search function is completed. The search function will be stopped by user whenever it returns the correct result. Search component has been optimized to reduce the amount of time that it takes to perform a search. However, it could consume more memory.

Figure 7-2 Value Panel



```
Result
Repository of 50 Components
  Component ID : 28@
    Maturity = 96
    cpu_utilization = 7
    Operability = 74
    Co_Existence = 12
    Testability = 13
    Adaptability = 49
    Satisfaction = 2
    Portability = 70
    Context_Coverage = 73
    Processing_Time = 144241
    Reliability = 57
    memory_utilization = 14
    Usability = 3
    Interoperability = 2
    Integrity = 95
    Fault_Tolerance = 10
    Appropriateness_Recognizability = 90
    Capacity = 74428683
    Throghput = 116148179
    Maintainability = 48
    Modifiability = 48
    Compatibility = 82
    Functional_Appropriateness = 8
    Reusability = 24
    Effectiveness = 15
    Performance_Efficiency = 59
    Response_Time = 224165
    Efficiency = 79
    Functional_Suitability = 52
```

Figure 7-3 Result Panel

The tools's interface is designed with 3 pages. when executing the application, the first page presented is the initial panel, as seen in Figure 7.1. This page allows the user to select a scenario among the available options and then choose a set of non-functional names and definitions that describe the quality of their desired component. The second page is the value panel, as seen in Figure 7.2. This page collects the user selection's values using text boxes and then submits them to the application. These values indicate the NFRs degree that the queried component should have. Moreover, it is able to validate user inputs against the non-functional defined valid ranges in the Section 6.5.1. If the inputs are not valid the user will not be able to submit the query. The third page is the result panel, as seen in Figure 7.3. This page represents the user query result. Moreover it allows user to re-run the application (by Start Over button).

The above Figures (Figures 7.1, 7.2 and 7.3) indicate an example of proposed prototype. Initial (Figure 7.1) and value (Figure 7.2) panels indicate that user quired a component with the following quality characters repository that consist of 50 component desctiption:

- **40% of Context Coverage:** a component that was useable in 40% of contexts that it has been tested
- **70% of Efficiency:** a component that used the required resurces efficiently in 70% of test cases while it was fully functional

The result panel (Figure 7.3) indicates the detail quality characteristics of selected component description. It consist of queried NFRs' prerequisites and (their prerequisites) such as:

- Usability: it is prerequisite of Context Coverage
    - Effectiveness: it is prerequisite of Usability
- Operability: it is prerequisite of Efficiency
    - Performance Efficiency: It is the prerequisite of Operability

The above result shows the ability of system to add prerequisites to refine the search automatically. Moreover, the system is able to add the prerequisites of NFRs' prerequisites whenever more than one component description matchs the user query to distinguishes between component description with similar scores.

A key research question in this work focuses on understanding whether semantic component indentification and selection (proposed method), i.e, is using a non-functional requirements ontology beneficial to the software development community. This question is addressed in the community case study discussed in Section 7.1.

Another research question addressed in this thesis is whether the use of a non-functional requirements ontology can support component identification. To answer this question, a repository is populated by a program which generates these component descriptions. The component descriptions are generated in the form of RDF files due to its machine readability. A mix of qualitative and quantitative analysis is performed on the data. Qualitative analysis helps to identify new knowledge and theories, while quantitative analysis helps to test the method's performance. These are discussed in Section 7.5.

7.1 Community Case Study

To address the questions related to the perception of non-functional requirements as well as test the tool within the software development community, a case study of component identification and selection is performed with a number of component search useage scenarios. The scenarios exemplify building an online store sucha as Amazon.com, and specifically, its advanced search function. A repository, containing 50 components descriptions, is used.

A qualitative study involving software engineers and developers is undertaken, where participants are able to use the tool and review it using an online questionnaire. All the participants were familiar component-based software development and requirement engineering process. An information sheet addressing the following questions is provided to participants. The questions are:

- What is the complexity of software development that we are working on and its solution?
- What is a software component and how and where to find it?
- What is the selection method based on?
- What are NFRs definitions?
- What is a scenario about?
- What is the user's input?
- How is the result presented?
- Who is undertaking the project?

Moreover, the participants were asked to undertake the following tasks:

- To read the information sheet.
- To run the tool five times.
- To complete the online questionnaire in the presence of the researcher (there is no verbal feedback).

Each session is expected to last for 30 minutes and the questionnaire takes approximately 15 minutes to complete.

Participants interacted with prototype interface by viewing a Graphical User Interface (GUI) that contained a list of NFRs name and list of scenarios. Whenever a participant selected an NFR name from the list based on a scenario, the NFR value of the selected one was collected and validated to search the repository.

The following scenarios are presented to the participants; each involves a major book retailer that utilizes a computer system to handle their online bookshop:

Scenario 1: The online bookshop has two types of users: website visitors and registered users who have been given different degree of page (data) access appropriate for their types. Visitors are limited to catalogues of books. However they are authorized to access the secure payment page when they become a potential buyer. The online bookshop provides a secure environment for buyers to use their credit card information. A login component provides the registration functionality which protects user information.

Scenario 2: Customers try to search for books on the website, either by author name, or by words in the title. A list of all matching books is returned to the customer. A customer does not need to be logged-in in order to search. The website is designed to provide accuracy and complete search functionality (search component). Once the user receives a result, satisgying the search criteria the search function is completed. The search function is stopped by the user whenever it returns the correct result. The search component is optimized to reduce the amount of time that it takes to perform a search. However, it could consume more memory.

Scenario 3: A sneak peek panel and a few buttons are added to the catalogue Section allowing customers to browse a few pages of the book they are interested to buy. This helps the buyer choose the most appropriate book according to their needs. Moreover, a user guide is provided to introduce this feature. To ensure good quality of this new component, website developers advance a survey-questionnaire to get users' feedback on their experience of using the website. The questionnaire helps to understand why people visit the site and whether the site meets the visitors' needs and expectations.

Scenario 4: Search component is designed by the bookshop developers. It is a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading. There is the possibility of reusing it in other projects which have different contexts. What is important here is measuring the possibility of reuse as the first step. There is a case with zero or small probability of reuse, but the search component can be modified, in this case, the second step is measuring the ease of modification. Developers may need to examine the internal capability of the search component to be customized by adding or removing search fields. The customization result shows the possibility of adapting the component to the new environment. The final step is testing the component to make sure the transfer to a new environment can be finalized, without negative impact on any other components or parts (resources).

Scenario 5: A Shopping Cart component includes a set of functions (steps) that help the user to buy a book. The component prompts for the customer's username and password. The customer enters these details. The component verifies the customer's identity and retrieves the customer's name and address, then prompts for credit card details. The customer enters these details. The component checks the credit card details. The component shows the customer the book and the delivery cost. The customer confirms the transaction. These sets of functions are designed in a way that meets users' needs and cover all the specified tasks and user objectives.

Scenario 6: The company development team have a clear idea regarding the type and amount of resources that is required for an online bookshop. Usually, this information is stated in a system requirement document. The following steps, describe parts of system memory usage without any specified degree of usage: 1-) the system records all books available in the online store, 2-) For each book, the author, title and ISBN number are recorded, 3-) The number of each book in stock is also stored, along with the number on order by customers and the number on order from publishers. All this information is retrievable by system functions.

Scenario 7: There exist a number of banking systems available for the online bookshop to facilitate its financial requirements. The current bookshop's banking channel is selected based on the following factors: 1-) Degree of which the system performs the banking functions under different network status and within time constraints, 2-) Degree of which the system avoids critical and serious failure occurrences against various failure patterns, 3-) Degree in which, the system in the event of an interruption or a failure, can recover the company or customers' data that is directly affected and re-establish the desired state of the system.

Scenario 8: The bookshop specialized support staff, operational staff or business staff might need to do some correction, improvement or adaptation on the their existing components due to changes in the environment or in the requirements and functional specifications based on the following factors that evaluates the fitness of each component: 1-) Degree of effectiveness and efficiency with which components can be modified, updated and upgraded without introducing defects or degrading existing quality, 2-) Degree of effectiveness and efficiency with which test criteria can be established for a single component and tests can be performed to determine whether those criteria are met.

The questionnaire plays an important role in gaining further knowledge of the users' needs. It also provides a tool for further analyses of the processes of selection, the scenarios, the non-functional requirements based descriptions, and the non-functional requirements values.

The survey in this study has three check boxes in the questionnaire and the reminder are in free-form text. The reason for this style of survey is to avoid possible bias as a result of guided responses through check box responses and to enrich the data collected. The including free text responces give users an opportunity to state, in their own words, what they have experienced, this often provides a richer account of the incident and the context in which the incident occurred.

We used a group post-test experimental design in order to minimize the internal threats to validity. The post-test is the set of questions that participants answered during the experiment. Testing, instrumentation and statistical regression internal threats are eliminated by not using a pre-test. The experiment did not need a control group and keeping the duration of the experiment to under 60 minutes eliminated the history and maturation threats. The two independent variables in this experiment are: the use of NFRs; and the use of a search tool based.

The experimenter was not able to eliminate all the external threats to validity. By conducting the experiment on the internet, we minimized the available participant threat to population validity. We believe that the following threats to validity were not a factor in our experiment: interaction of history and pretest-post-test sensitization.

The case study involved thirty participants who have programming experience, ranging from beginners to experts. Information sheets are provided to reqruite the case study participants. Those sheets allowed them to initiate contact about the study. The expertise level is measured by the number of years they were involved in software development. Participants with 1 to 5 years experience are considered beginners and those with 6 years or more were considered as experts. Out of 30 participants, 28 only specified their year of experience. The results indicated that 31% of participants are experts and 68% are beginners. Thematic analysis is used with the open ended responses. The systematic analysis resultes in a few key findings, discussed below in Section 7.2.1, 7.31 and 7.4.1.

7.2 Research Question: What type of component description express all of the information that is needed by the community?

Most component descriptions focus on the functionality of components. The aim of this question is to obtain the opinion of experts about the use of descriptions for non-functional requirements. To address this question, the user's opinion of utilizing non-functional requirements description is studied using the following questions:

- What do you understand non-functional requirement-based component descriptions to be?
- What type of component description do you prefer to use? Why?

The first question was in free text form. The first part of the second question is in check box form, providing answers of ‘functional requirement base’, ‘non-functional requirement base’ and ‘other’. In the other option, however there is a free text box where respondant can provide further information. The second part of the question i.e. ‘why?’ has a free form text option for the respondant.

Thematic analysis is used to process the qualitative information obtained in this Section. A list of themes or patterns in the participants’ responses are defined. Then, the number of times these qualitative themes occurs is counted. This process allows for the qualitative information to be translated into quantitative data.

7.2.1 Results

Generally, components are specified by their functional and non-functional capabilities or descriptions (defined in Chapter 2). This Section summarizes the participants’ point of view regarding these descriptions.

In regards to the description definition, 3 themes in the participants’ responses are identified. These themes are counted as follows: 53% users (16 of the 30 users) define the elements in the description as quality attributes and 23% (7 of 30 users) defined them as non-functional names and values for a specific component. Therefore, 76% of users (23 of the 30 users) understood this type of description based on non-functional requirement-based component definition in Chapter 2.

The third theme includes 23% of participants, (7 of 30) which they did not provid related respond to the questions. However, they defined a non-functional requirements based component description as 1-) additional information to describe the components, 2-) the degree to which one component can fit the specified requirements, 3-) just simply same list of selected requirements that had been chosen to select the best component.

For example: one user understanding of non-functional requirments is "A summary that shows to what extent a particular function matches the non-functional requirements that I asked for, and that includes additional useful information about the properties of the function"

There are three main responses regarding the preferred type component description. Firstly, the majority of participants prefer to have both non-functional requirements and functional requirements based component descriptions to achieve the best result of reuse. In this case a user commented that:

"*For the software development process we need to identify both functional and non-functional requirements very precisely to achieve the expected goal (through well-defined functional requirements) and the overall quality (through a rich set of non-functional requirements descriptions like the ones provided by this tool)*".

Secondly, 16% of users (5 of 30) prefer non-functional requirements in order to have additional information about their selected component. A user comments:

"Functional requirements can be explained better in text descriptions, but non-functional requirements are good for a search tool such as this, as they can be used to compare many functionally similar components".

Moreover, another user comments:

"Both are equally important - however, the non-functional requirement's present options to interpret the role and associations between functional and non-functional requirement's relevant to the actual functionality".

Thirdly, 23% (7 of 30) of users emphasized having a functional requirement based component description due to their ease of use and understandability. In the survey a user comments that:

> *"I feel that as a programmer I would prefer a functional requirement base, but if I was to be a System Architect or equivalent, having a non-functional requirement base would be more useful."*

The survey result demonstrates that almost one quarter of users do not have a clear idea about non-functional descriptions. Moreover, those familiar with non-functional requirements prefer to simultanueosly have functional requirements.

7.3 Research Question: Is our approach useful for the community?

Requirements analysis and description analysis are among the key processes of component selection. The aim of this question is to understand what the users perceive as the advantages and disadvantages of this approach. To address this question, the user's overall experience is investigated by asking the following questions, which were all in free form text with the exception of the final question (explained later):

- If we had to build this tool again, what would you change in terms of processes?
- What are the things that you like most about this tool?
- What benefits do you expect to see from using a tool like this?
- What difficulties did you have when using this tool?
- What was your overall experience in using this tool?
- When you think about this new tool, do you think of it as something developers might need or as something developers might want? Why?

The first part of the last question is in check box form and the second part of the question i.e. 'why?' had a free form text option for the response.

Thematic analysis is used to process the qualitative information obtained in this Section. Results are presented in the quantitative format.

7.3.1 Results

The user responses indicate that this tool increases their non-functional requirements awareness. The participants believe that the tool reduces the time of component selection and consequently the total time of software development. Moreover, they have positive experiences and views on tool components (name selection, value selection, validation, data submission and result presentation) based on their response to the survey questions; they referred to the tool as interesting or good. However, there is some rooms for improvement. Firstly, users prefer to deal with simplified processes (Name and Value selection). Their main difficulty is choosing the non-functional requirement names and values. While the tool has provided the non-functional requirements definition and the accepted ranges, users found the non-functional ranges difficult to visualise and the definitions difficult to understand. Moreover, the tool is designed to present the best result according to the query submitted but users also expressed an interest in the second and further matches. The results are summerized below:

The user responses suggest a few changes or improvements of the tool's processes and GUI are warranted. For example, a user suggests that:

"Making the scenario box wider."

The user also preferred having check boxes instead of multiple item selection by using the control key. Moreover, a user suggested that

"Listing all the non-functional requirements without a scroll bar."

The remainder of the user requests for improvement are in terms of non-functional requirements range presentation, functional based scenarios, having an alternative result and using a bigger repository. A user comments that:

“It is hard to visualize the range for non-functional requirements. Maybe it could show some examples.”

Participants indicate that the non-functional requirements are choosen easily if the tool could use a priority based search technique. Some users comment that:

“Seems like a good idea, but doesn't have a flexible priority based search when one or more non-functional requirements are missing from the component.”

“Classify non-functional requirements in some way to make easy to select them. Maybe some class of hierarchically organization.”

“Providing numbers for non-functional requirements. It's better to allow users to prioritize them.”

In regards to the benefits that users can obtain from this tool, 33% (10 of 30) of participants believe that the tool facilitates software development, especially CBD in terms of time and cost of development. Moreover, users indicated that the tool increases awareness of non-functional requirements, therefore, reducing the time needed for software development. A user commented that:

“Reusable code is so great and such a tool to find the match saves a lot of time”.

The participants overall experience of using this tool indicates that it helps to reuse software components and it improves the developers’ productivity. However, it is good for developers who do not have much experience. More than half (53%, 16 of 30) of the participants agreed that the tool is simple to use and it speeds up development. One of the users comments that:

"I think that some users, who want to get straight into the task of programming would enjoy having a component provided which would enable them to program the business logic more quickly. Other users which are not as experienced may benefit from using this as it provides defined components, which the user may not be aware of."

Most of the participant's difficulties related to dealing with non-functional requirements. Firstly, 36% (11 of 30) of the participants said that they had difficulty using the tool due to their lack of non-functional requirements knowledge. Secondly, 26% (8 of 30) of users had difficulty in choosing the right value and visualizing the ranges. Thirdly, 16% (5 of 30) users do not expect to see a long list of non-functional requirements. Finaly, 3% (1 of 30) of users had difficulty to understand the scenarios. In comparison, the number of people who had difficulty with understanding non-functional requirements descriptions were 10% more than those that had difficulty choosing the values. A few users commented that:

"So many choices. Grouping them would help user to select things easier".

"It is hard to visualize the range for non-functional requirements. Maybe it could show some examples."

"It can provide not only the non-functional requirements I have thought relevant but also the affect from many other non-functional requirements that could affect, which I have not foreseen."

The validation feature that checks the submitted value(s) and the full list of non-functional requirements obtained positive responses from participants. Two users comment that:

"Values enable one to better control the performance of the software."

“It highlights any error. It specifies all the values and NFRs in relation to the selected ones.”

Questions relating to needing or wanting this tool gained positive responses from the participants.. Firstly, 40% (12 of 30) of participants said that they “need” this tool and, 36% (11 of 30) said that this tool is something developers “want” for their software development activities”. Secondly, 12% (4 of 30) of participants said that developers both ‘need’ and ‘want’ this tool. One of the users states:

“This tool would not only be beneficial for developers but it would also be a great tool for stakeholders of software design and development process.”

6% (2 of 30) users stated that developers’ feelings (“want” or “need”) are dependent on the complexity of the components repository. This could be because of the current trend of using pre-built templates (Twitter Bootstrap [128], Rails Scaffolds [129] etc.) as well as the current surge in component based design (HTML 5 Components [130], Google Polymer [131]). However, selecting the correct component among a large set of available components is a difficult task.

The participants answered the benefits of using the tool in a free text form. The responses are analysed using thematic analysis. The answers were categorized in Table 7-1:

Themes	Number of Participants
The tool helps participants **discover potential alternatives that they have not thought of**	6% (2 of 30)
Most of the developers think of the functionality of components	6% (2 of 30)
The tool helps to select the correct component among a large set of available components	20% (6 of 30)
The tool improves non-functional requirement awareness in developing software	16% (5 of 30)
The tool makes software development **easier**	13% (4 of 30)
The tool helps participants **save time**	36%(11 of 30)

Table 7-1 Tool Benefits for Participants

According to the collected user feedback, the tool in this study is capable of facilitating the software development. However, developers have difficulty to shift from traditional functional selection to non-functional requirement-based selection. They have difficulty in analysing the scenario and in picking the required non-functional requirements. A user comments:

"Sometimes these non-functional requirements are vague and I prefer to corporate some functional requirements to clear them more."

Another user comments:

"Some of the terms that were used as non-functional requirements were quite difficult to understand."

Moreover, another user comments:

"Some functional requirements are easy to recognize from the description directly by human while some are hard to analyse and hard to get reliable results by machine. However, non-functional requirements can be discovered by machine analysis based on

previous experiences; but they hide in subtle patterns of the requirements and are hard to discover by human beings."

7.4 Research Question: What does the community understand non-functional requirements to be?

The aim of this question is to investigate the way users distinguish between functional requirements and non-functional requirements. The users' knowledge and experience are investigated by asking the following questions:

- What do you understand non-functional requirements to be?
- Do you think values are necessary when specifying non-functional requirements or simply selection of non-functional requirements is sufficient?

The first question was in free text form. The second question was in check box form with an option of 'other' in free text form. The answers from all thirty participants are analysed using thematic analysis format for the free text form survey questions. Results are explained in the following Section.

7.4.1 Results

This Section is divided into two parts: 1-) focusing on the way users deal with non-functional requirements when they want to elicit them from scenarios and 2-) on how users deal with non-functional requirements when they want to combine them to search for a desired component. The results indicate that users are aware of the importance of non-functional requirements but they are not able to pick the right ones due to lack of non-functional knowledge.

First of all, 20% (6 of 30) of participants did not have a good understanding of the non-functional requirements. They believed non-functional requirements are:

“Properties that are not related to functional requirements”

“The things we expect a software program to have (the "Nice-things" it can do) except for real functionality”

“It is not related to the actual program function but are more descriptive of the environment it performs in”

Generally, the majority of participants believe that the non-functional requirements are difficult to understand but they are necessary for quality and success of development. Firstly, 43% (13 of 30) of participants believe non-functional requirements help software components to achieve their goals. Secondly, 26% (8 of 30) of participant indicate that it is necessary for components to have non-functional requirements and define them as something related to quality.

The purpose of the “value selection process” is to provide each non-functional requirement with a range. Thus, users are able to specify the degree of each non-functional requirement. This process is viewed positively by 70% (21 of 30) of participants. They believe that non-functional requirements’ value is necessary for component selection.

According to the feedback, developers are not familiar with the definition of non-functional requirements. However, they have general knowledge of software quality and its importance. The scenarios reveal the following:

An understanding of non-functional requirements definitions in necessary. Users only need to select the non-functional requirements semantically highlighted in the scenario and then the tool task is to improve user’s query by adding the important non-functional requirements. Based on the analysis 73% (22 of 30) of participants believe that the scenarios are close to real world software requirements. However, 20% (6 of 30) of the participants had difficulty in prioritizing the non-functional requirements. Moreover, the time that participants spent to analyse each scenario; 63% (19 of 30) of them spent less than six minutes on each scenario. A participant commented that:

> *"I prefer to have some hints for selecting best components based on the given scenario and quality attributes."*

7.4.2 In Summary

The community case study demonstrated that, developers are not familiar with NFR terms and NFR based descriptions. However, they believe NFRs are essential for investigating how a component performs. The NFR knowledge provided by the tool is considered valuable but not critical to project success and yet the participants would like to have the functional requirement descriptions at the same time. Thirdly, participants liked the idea of specifying a degree for NFRs but the analysis shows that they have difficulty in deciding on and selecting, the degree to which the NFRs must be met. Finally, participants believe this approach increases the developers' productivity in terms of search , and the quality of the results identified.

The above findings confirm that the software community has a generic idea regarding NFRs but not sufficiently deep to be useful. Moreover, the combination of FR and NFR-based descriptions is preferable to the participants. The above findings confirm that the approach is useful for the community in terms of: developers' productivity, time of development, NFR awareness and ease of selection.

7.5 Performance Analysis

7.5.1 Runtime Analysis

In order to evaluate the efficiency of the approach used whitin thesis, the runtime of all possible query types is examined. Two types of sampling are employed for testing the validity of the approach's performance, namely, probability and non-probability sampling [59]. For this purpose, the following runtime tests are performed:

- Runtimes of individual non-functional requirements against various size of repositories (Figure 7.4)
- Runtimes of various queries of difficulty level from basic to complex against various size of repositories (Figure 7.5)
- Runtimes of random queries against various size of repositories (Figures: 7.7 and 7.8)

Component repositories of 50, 100, 200, 400, 600, 800, and 1000 component descriptions are built to verify the approach using a Java application. This application generates RDF statements (component description). Each non-functional requirement runtime depends on 3 factors: 1-) number of prerequisites for that non-functional requirement, 2-) the content of component descriptions and 3-) the result.

Moreover, the runtimes are an average of 10 runs and the PC OS specification for this experiment was an Intel Core i5 CPU, 2.5 GHz and with 2GB of RAM.

7.5.1.1 Prerequisite dependency

Each non-functional requirement has between 0 to 6 prerequisites. As mentioned earlier (Section 6.3.2), the non-functional requirements based search adds each element's prerequisites to the query. For a non-functional requirement with a larger number of prerequisites, the algorithm searches for more items and therefore, the runtime is higher (Figure 7.4).

7.5.1.2 Component description dependency

The weight assignment strategy works based on the rules described in Chapter 5. It maps the calculated score to each component as a comparison factor in the algorithm. The rules of the weight assignment strategy are based on the content of a description. For a component with a larger number of non-functional requirements a longer analysis time is required. Moreover, the algorithm reduces the search space by eliminating the components that do not satisfy the user query for further analysis of query prerequisites.

7.5.1.3 Result dependency

When the system find more than one component with the same weight, the algorithm overrides the query by calculating and adding prerequisites of the previous set of query element to the existing ones and does the overall process of assessment again.

Figure 7.4 bar charts compare the list of 32 non-functional requirements against 7 different repositories in terms of size and content similarity. Overall, the size of the repository has a direct effect on the length of runtime. Pursuing this further, the runtime of most non-functional requirements against different repositories showed similar patterns, with all gradually increasing at a steady rate from 50 components repository to 1,000 components repository. However, the runtime of some non-functional requirements against 800 components repository became significantly higher than the runtime of same non-functional requirements against largest repository (1,000 components repository). This happens when the requested non-functional requirements appear in more components of the smaller repository (800 components repository). Likewise, some bars of the largest repository have sharp peaks (not increasing at a steady rate) in comparision with other bars' normal rate. By looking at operability, context coverage and performance efficiency as rich non-functional requirements in term of prerequisites (5 to 6 prerequisites), it is found that as the number of prerequisites increases the runtime of non-functional requirements increases as shown in Figure 7.4. The runtime of these non-functional requirements are calculated based on 5 or 6 prerequisites.

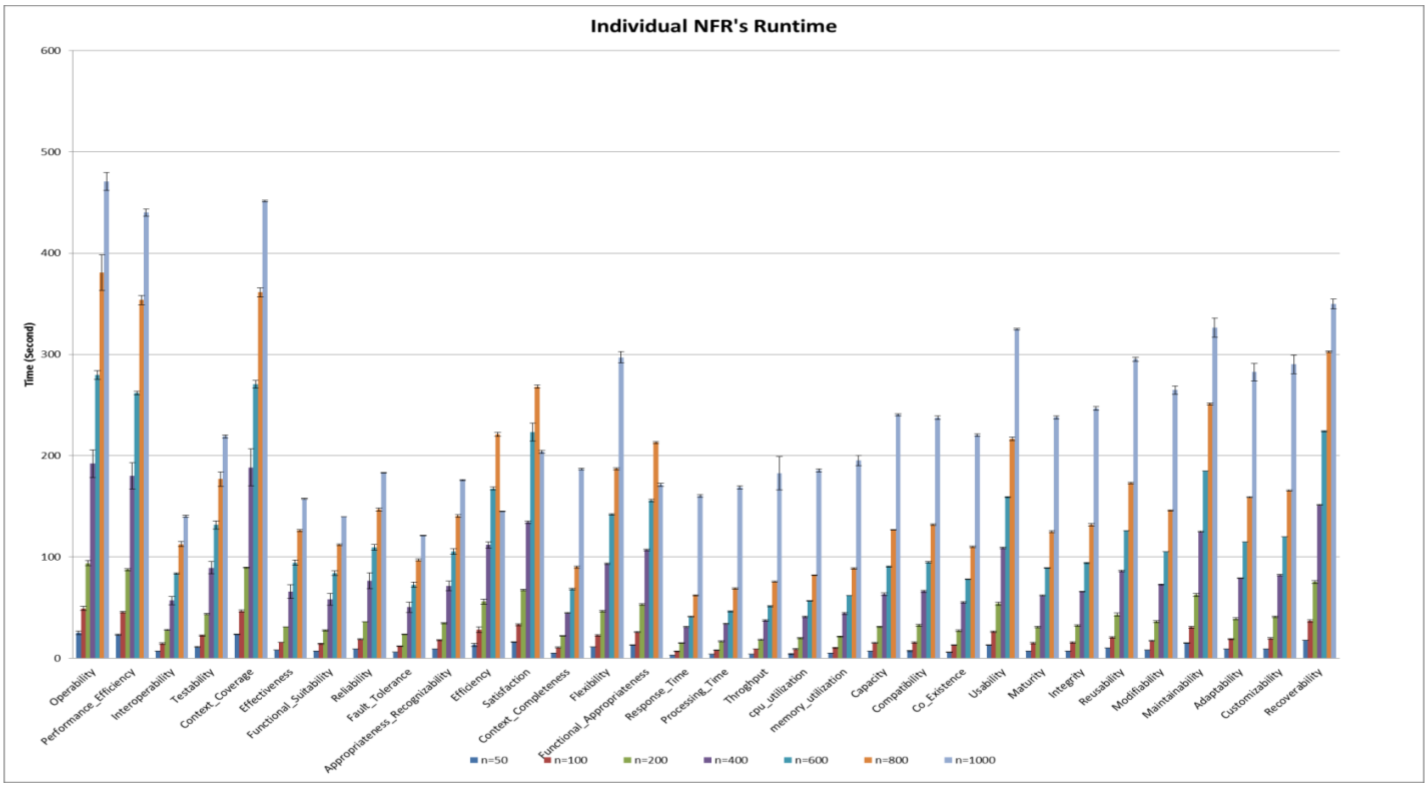


Figure 7-4 Runtime for single Non-functional requirements queries with repositories of various sizes

Figure 7.5 charts compare 8 different groups of non-functional requirements in terms of size and NFRs' complexity against 7 different repositories in terms of size and content similarity. Overall, the runtime is longer whenever an extra non-functional requirement is added to a group. Groups of non-functional requirements are selected in order to obtain some basic and complex queries. Based on the graph for each non-functional requirement group,

group 5 showed a runtime increase due to the complexity of the query. This group contains 5 non-functional requirements such as portability, performance efficiency and context coverage. The total prerequisites' number for this group is 16, which makes the query more complex to analyse.

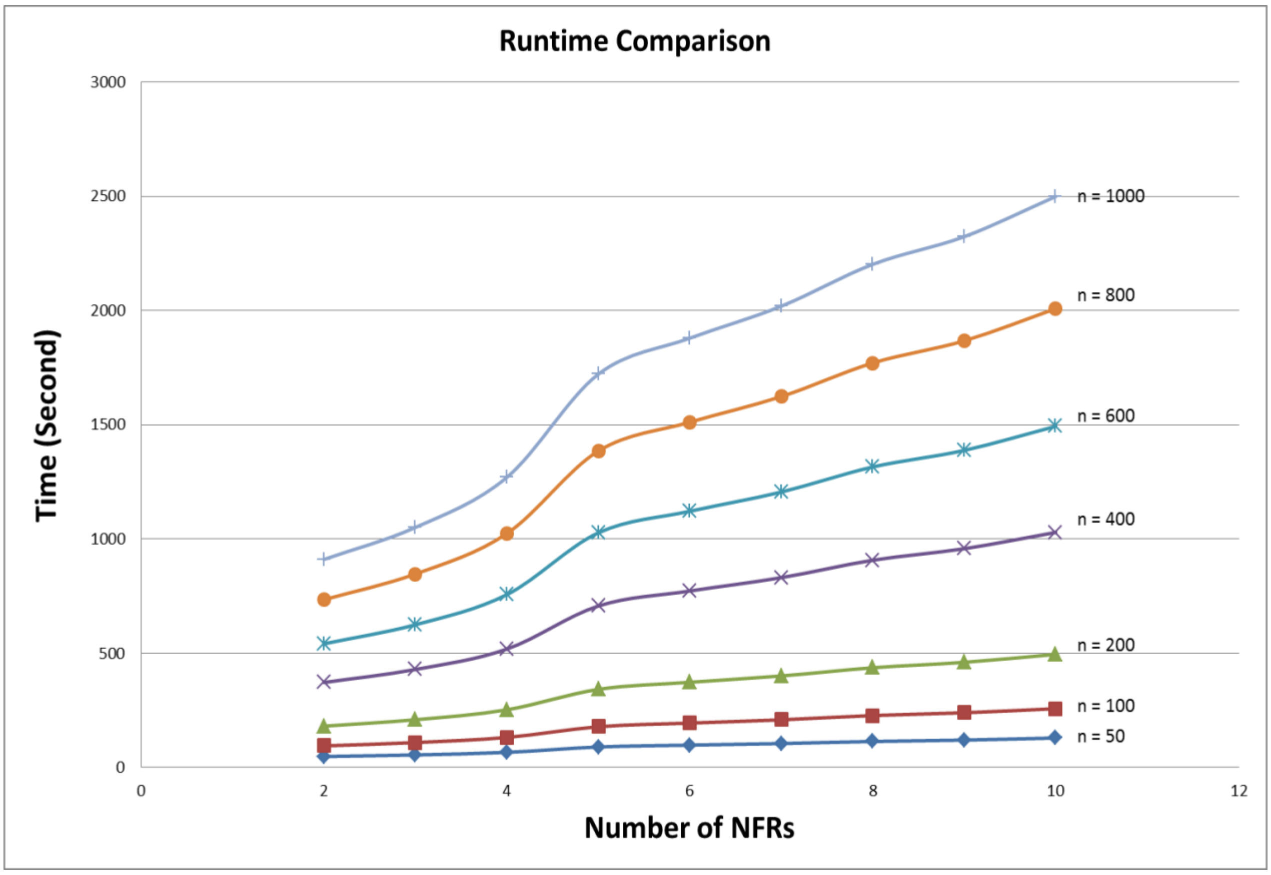


Figure 7-5 Runtime for predefined grouped Non-functional requirements queries with repositories of various sizes

Figures 7.7 and 7.8 compare 8 different sized groups, which contain random members against 7 different repositories in terms of size and content similarity. Overall, the runtime of most groups against different repositories showed similar patterns, with all gradually increasing at a steady rate from a 50 component repository to a 1000 component repository. However, two changes from group 3 to 4 and group 5 to 6 are shown due to the distribution of non-functional requirements with a higher number of relationships in groups 4 and 6. Moreover, both charts show that the runtime of groups 4 and 5 are almost similar. However, group 5 has one more non-functional requirements to analyse but fewer queries to analyse and calculate. According to the rules defined in Chapter 5, for the purpose of weight assignment, the algorithm needs to check the type of concept (super and sub concepts) for each non-functional requirement. A set that contains fewer super concepts requires less analysis and calculation. The following sample checks the type of Functional Suitability:

```
"SELECT  ?super WHERE { "
"BIND(   ont:"functional_suitability"   as
concept )"
 "?concept rdfs:subClassOf ?super ."
 "OPTIONAL{"
 "?concept rdfs:subClassOf ?inbetweener ."
 "?inbetweener rdfs:subClassOf ?super . "
"FILTER(?inbetweener            !=?concept
&&?inbetweener      !=      ?super      &&
!isBlank(?super))"}"
"FILTER(!BOUND(?inbetweener) && ?super !=
?concept && !isBlank(?super))"}"
```

Figure 7-6 Sample query to check the type of NFR

Both Figures (7.7 and 7.8) indicate that the sampling was random.

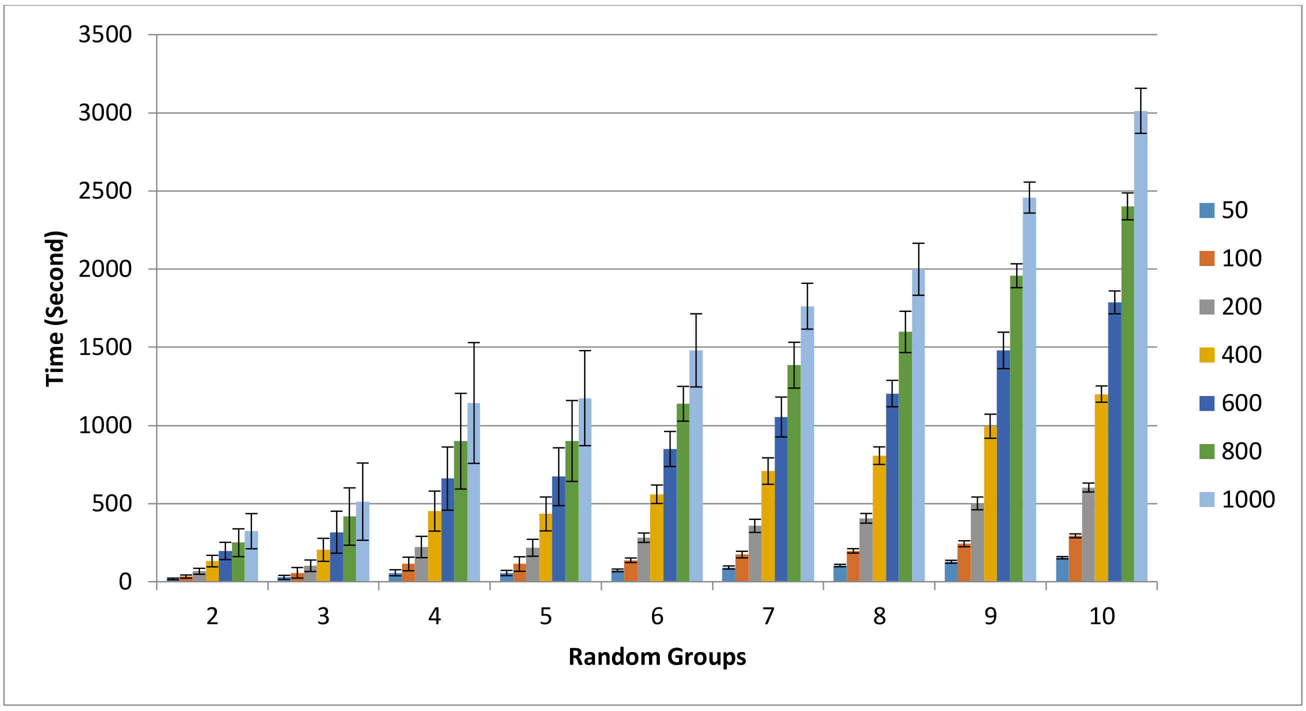


Figure 7-7 Runtime for random grouped Non-functional requirements queries with repositories of various sizes

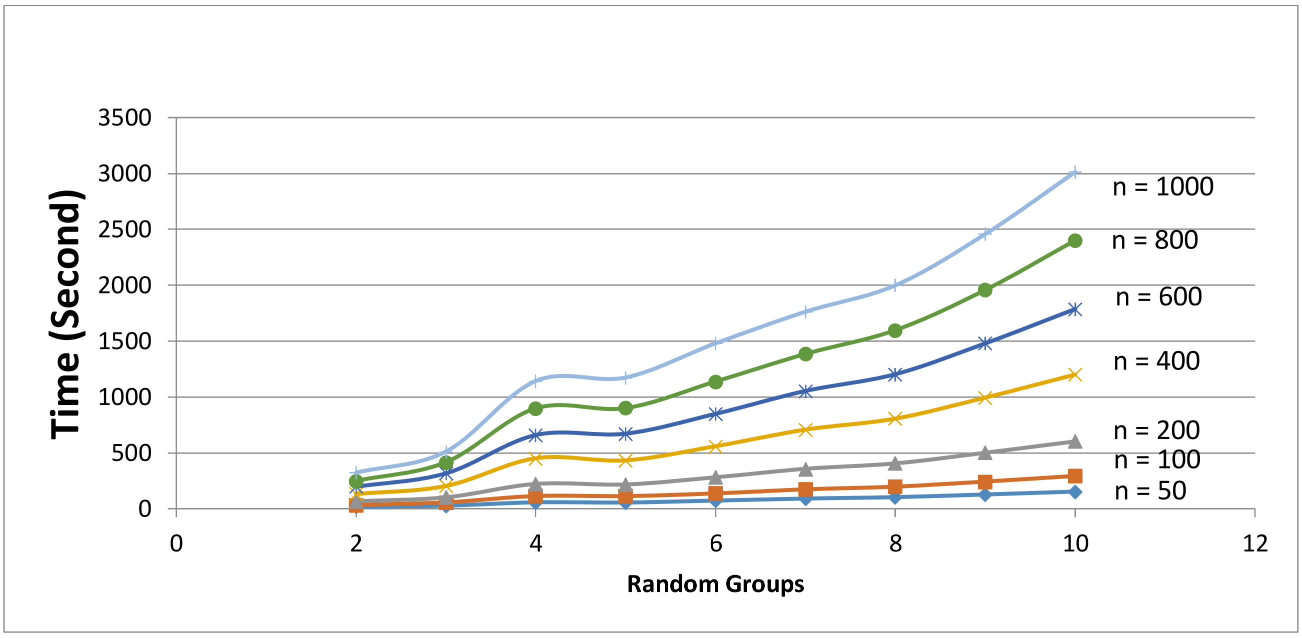


Figure 7-8 Runtime for random grouped Non-functional requirements queries with repositories of various sizes

One of the findings of this study is that the complexity of NFRs depends on the number of operations (defined rules to be checked in order to extract knowledge), which are required for their analysis. The main operation is the interdependency check among NFRs, which requires identification of their prerequisites. As the number of computation increases, the runtime increases.

Moreover, based on this study, the runtime on each NFR (evaluation) depends on its meaning. The more queries sent to the ontology, the more time requires to complete the computation. This is as a result of having a single thread of execution, where all rules and relations were computed one-by-one to generate a score. This score was used in retrieving the best match. This limitation can be addressed in future by using multi thread of execution. This will be much more efficient and can cope with the complexity of NFRs analysis.

7.5.2 Summary

Participant feedback and runtime comparisons of non-functional requirements are presented in this chapter. The aim of the experiments is to analyse the usability of the tool based on the software developers' views. However, the usability of the algorithm is tested by running various queries (simple, complex) against different repositories. The findings of this approach are limited by the random component descriptions. Using real component descriptions that contain pair of non-functional requirements names/values is out of the scope of this study. The distribution of non-functional requirements, their quantity and values (among defined ranges) are completely random. The process for the experiential tool increased the non-functional requirements awareness among users. A new format for a component description (non-functional requirement / quality attributes and their values) is introduced. To overcome the complexities of non-functional requirements, the manual reasoning is transformed to the automatic reasoning.

8 Conclusions and Future Work

The analysis demonstrates the utility of the approach as a successful use of ontologies to enhance component selection. However, there is still room for further work. This chapter discusses the contributions and possible extensions of this work. In this thesis, the new concepts and background definitions discussed in Chapter 1 and 2 provide the evidence for the absence of a similar approach through the literature review in Chapter 3. Having then identified the challenges that face the component selector in Chapter 4, a methodology is developed in Chapter 5 to address the identified challenges. This leads to details of implementation and design in Chapter 6 and, description of the experiments and discussion of results in Chapter 7. This chapter explores how the approach can be extended and summarizes the key conclusions of the work.

The retrieval of software components is a fundamental issue within the component-based software development (CBSD) where components are retrieved for a system to build by locating (identifying) existing components in the repository. Developers mainly search for components based on their functional requirements, and less so based on non-functional requirements. This is due to the difficulty of identifying and analysing the non-functional requirements. With regards to functional requirements, there are different frameworks that automated component identification, using domain knowledge ontologies. With regards to non-functional requirements, there are manual or semi-automated approaches that introduce a group of techniques to analyse and choose non-functional requirements correctly. Though they are sufficient for non-functional requirements analysis, they are expensive to perform. In order to reduce the cost of non-functional requirements analysis, an automated component selection mechanism is required, which provides the non-functional requirements knowledge and definitions.

This research studies issues such as a review of the quality models from the quality relationship and definition perspective. Moreover, component retrieval techniques from component search and specification perspective are reviewed. This research also includes the study of techniques to address how component software requirements knowledge can be shared and reused.

8.1 Contribution

This thesis has documented the development of an automated component selection framework based on non-functional requirements. To analyse non-functional requirements, this framework is powered by an ontology of 32 non-functional requirements that each might have among 0 to 6 prerequisites. The ontology is designed to share the non-functional requirements knowledge among different processes such as query analysis, query reformulation and component description analysis. The knowledge is used for interpreting different properties of non-functional requirements individually or in comparison with others. This study demonstrates that non-functional requirements based ontology supports component identification. Reuse of this ontology in the functional based approaches can augment the functional aspect of the search in order to reduce the number of potential components identified. Previously, the non-functional requirements knowledge produced through a manual or semi-automated techniques, such as group discussion, questionnaires and brainstorming. Those techniques were costly and complex.

The demonstrated framework is unique in that its input and output are constructed based on a set of non-functional requirements names and values, which are based on the quality attributes of the system features. In previous works, queries and component specifications are constructed based only on functional requirements i.e. input-output relation, natural language, pre-conditions / post-condition, component profile, conceptual graph and mathematical analysis. The framework used in this study results in the reduction of inconsistency and in reducing incompleteness in requirements specification. Therefore, it supports the component's elicitation based on non-functional requirements specification.

The findings demonstrate the benefit obtained from using an ontology by minimizing the cost and complexity of analysing non-functional requirements. The algorithm runs a complex query that has 5 non-functional requirements with a total 16 prerequisites against a repository of 1000 components in 1750 seconds. It is impossible for a field expert to compute a complex query in a this amount of time. However, 20% of users had difficulty in choosing the required set of non-functional requirements from the high satisfactory scenarios (90%). Moreover, the ontology addresses the complexity of the requirement descriptions in a way that the missing prerequisites automatically are added to the query. 75% of participants only have a clear idea about non-functional descriptions and this group prefers to have functional descriptions at the same time to facilitate their component selection. Furthermore, the runtime of the system, when selecting components based on individual non-functional requirements, is reduced by using an ontology. This improves the efficiency of the resultant system. Lastly, 75% of participants mentioned that there is a need for such a tool in the software development community.

8.2 Future work

The component description model used in this study consists of two elements, namely, the non-functional requirements names and values. To evaluate the number of non-functional requirements in a description and assign a value to them, a testing framework is needed for non-functional requriremtns mesurments. This framework should be able to test a software component and provide a machine-readable description file. The file should contain the component's non-functional requirements and their calculated values. In order to evaluate a component's non-functional requirements or quality, the testing framework needs to be equipped with component based metrics and measurement methods. In the introduction (Chapter 1) and methodology chapters (Chapter 6), the way that component descriptions are generated for the purpose of this study is discussed. A testing framework design and development would be one of the extensions of this study. The ISO 25020 and 25021 standards [47, 48] provide some recommended quality measures for software development and testing. Moreover, the ISO standard provide some metrics and measurement methods for general use that they are not suitable for CBD. Thus, a testing framework is required to design, implement and evaluate the component-based interpretation and validation. The testing framework may be able to predict or estimate the non-functional requirements. So, this possibility needs to be investigated. The prediction (of NFRs) would be possible when the framework uses data related to non-functional requirements. On the other hand, estimation (of NFRs) would be possible when the framework uses facts about non-functional requirements, such as interdependencies among them.

In conclusion, this thesis shows that the use of component selection tools with a focus on non-functional requirements can benefit from an ontological representation of the non-functional requirements. The ontology model is domain-independent, defined through a process of analysis, and customization of existing quality models. This study suggests the use of domain-independent mechanisms to support ease of component selection across domains, supported by domain-specific mechanisms where necessary, to assist in refining the selection process.

References

1. Wnuk, K., B. Regnell, and B. Berenbach, *Scaling up requirements engineering–exploring the challenges of increasing size and complexity in market-driven software development*, in *Requirements Engineering: Foundation for Software Quality*. 2011, Springer. p. 54-59.

2. Irshad, M., et al. *Capturing cost avoidance through reuse: systematic literature review and industrial evaluation*. in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. 2016. ACM. p. 35.

3. Sametinger, J., *Software engineering with reusable components*. 1997: Springer Science & Business Media.

4. Varnell-Sarjeant, J., A.A. Andrews, and A. Stefik. *Comparing reuse strategies: An empirical evaluation of developer views*. in *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*. 2014. IEEE. p. 498-503.

5. Mohagheghi, P. and R. Conradi, *Quality, productivity and economic benefits of software reuse: A review of industrial studies.* Empirical Software Engineering, 2007. **12**(5): p. 471-516.

6. Doerr, J., et al. *Non-functional requirements in industry-three case studies adopting an experience-based NFR method*. in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. 2005. IEEE. p. 373-382.

7. Chung, L. and K. Cooper, *Defining goals in a COTS-aware requirements engineering approach.* Systems Engineering, 2004. **7**(1): p. 61-83.

8. Mylopoulos, J., L. Chung, and B. Nixon, *Representing and using nonfunctional requirements: A process-oriented approach.* IEEE Transactions on Software Engineering, 1992. **18**(6): p. 483-497.

9. Franch, X. and J.P. Carvallo, *Using quality models in software package selection.* Software, IEEE, 2003. **20**(1): p. 34-41.

10. Wagner, C., *Model-driven software migration: A methodology: Reengineering, recovery and modernization of legacy systems*. 2014: Springer Science & Business Media.

11. Kay, R., *QuickStudy: System development life cycle.* Computerworld, 2002. **14**: p. 18.

12. Radack, S., *The system development life cycle (sdlc).* Computer Security Division Information Technology Laboratory 2009.

13. *Control and Audit, Information Systems. SDLC.* Institute Of Chartered Accountant Of India, 2013: p. 5.28.

14. Krueger, C.W., *Towards a taxonomy for software product lines.* Software Product-Family Engineering, 2004: p. 323-331.

15. Robey, D., R. Welke, and D. Turk, *Traditional, iterative, and component-based development: A social analysis of software development paradigms.* Inf. Technol. and Management, 2001. **2**(1): p. 53-70.

16. Voas, J., *Maintaining component-based systems.* IEEE Software., 1998. **15**(4): p. 22-27.

17. Krueger, C., *Easing the transition to software mass customization*, in *Software Product-Family Engineering*. 2002, Springer. p. 282-293.

18. Tiwari, A. and P.S. Chakraborty. *Software component quality characteristics model for component based software engineering*. in *Computational Intelligence & Communication Technology (CICT), 2015 IEEE International Conference on*. 2015. IEEE. p. 47-51.

19. Iribarne, L. and A. Vallecillo. *Searching and matching software components with multiple interfaces*. in *Proc. of the TOOLS Europe'2000 Workshop on Component-Based Development*. 2000.

20. Barros-Justo, J.L., et al., *What software reuse benefits have been transferred to the industry? A systematic mapping study.* Information and Software Technology, 2018.

21. Mahmood, S., R. Lai, and Y.S. Kim, *Survey of component-based software development.* Software, The Institution of Engineering and Technology (IET), 2007. **1**(2): p. 57-66.

22. Crnkovic, I., M. Chaudron, and S. Larsson. *Component-based development process and component lifecycle*. 2006. IEEE Computer Society, Proceedings of the International Conference on Software Engineering Advances. p. 44.

23. Crnkovic, I. *Component-based software engineering for embedded systems*. in *Proceedings of the 27th international conference on Software engineering*. 2005. ACM. p. 712-713.

24. Gan, G.G.G., W.K. Yen, and M. Toleman, *Software component reuse in information systems development: a review of challenges and strategies.* Han Chiang College, 2005. **3**: p. 83-95.

25. Sharma, S. and P. Shirisha, *An Intelligible Representation Method For Software Reusable Components*, in *Innovations and Advances in Computer Sciences and Engineering*. 2010, Springer. p. 221-225.

26. Prieto-Diaz, R., *Implementing faceted classification for software reuse.* Communications of the ACM, 1991. **34**(5): p. 88-97.

27. Ravichandran, T., *Special issue on component-based software development.* ACM SIGMIS Database, 2003. **34**(4): p. 45-46.

28. Prieto-Diaz, G.J.a.R. *Building and managing software libraries*. in *COMPSAC 88*. 1998. Chicago, IL, USA IEEE Xplore. p. 228-236

.

29. Gill, N.S., *Reusability issues in component-based development.* ACM SIGSOFT Software Engineering Notes, 2003. **28**(4): p. 4-4.

30. Frakes, *Software reuse as industrial experience.* American Programmer, 1993. **6**(9): p. 27-33.

31. Frakes, W.B. and T.P. Pole, *An empirical study of representation methods for reusable software components.* IEEE Transactions on Software Engineering, 1994. **20**(8): p. 617-630.

32. Alves, C. and J. Castro. *CRE: A systematic method for COTS components selection*. in *XV Brazilian Symposium on Software Engineering (SBES)*. 2001. Rio de Janeiro, Brazil.

33. Zulzalil, H., et al., *A case study to identify quality attributes relationships for web-based applications.* IJCSNS, 2008. **8**(11): p. 215.

34. Kaur, K. and H. Singh, *Candidate process models for component based software development.* Journal of Software Engineering, 2010. **4**(1): p. 16-29.

35. Wiegers, K.E., *Software requirements*. 2009: O'Reilly.

36. Andrew Stellman, J.G., *Applied Software Project Management*. 2005: O'Reilly.

37. Chung, L. and J. do Prado Leite, *On non-functional requirements in software engineering.* Conceptual modeling: Foundations and applications, 2009: p. 363-379.

38. Berander, P., et al., *Software quality attributes and trade-offs.* Blekinge Institute of Technology, 2005.

39. Rahman, M. and S. Ripon, *Elicitation and modeling non-functional requirements-a pos case study.* arXiv preprint:1403.1936, 2014.

40. Zowghi, D. and C. Coulin, *Requirements elicitation: A survey of techniques, approaches, and tools*, in *Engineering and managing software requirements*. 2005, Springer. p. 19-46.

41. Mylopoulos, J., L. Chung, and E. Yu, *From object-oriented to goal-oriented requirements analysis.* Communications of the ACM, 1999. **42**(1): p. 31-37.

42. Nixon, B.A., *Management of performance requirements for information systems.* IEEE Transactions on Software Engineering, 2000. **26**(12): p. 1122-1146.

43. Chung, L., B.A. Nixon, and E. Yu. *Using non-functional requirements to systematically support change*. in *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*. 1995. IEEE. p. 132-139.

44. Berenbach, B. and M. Gall. *Toward a unified model for requirements engineering*. in *Global Software Engineering, 2006. ICGSE'06. International Conference on*. 2006. IEEE. p. 237-238.

45. Morisio, M., M. Ezran, and C. Tully, *Success and failure factors in software reuse.* IEEE Transactions on software engineering, 2002. **28**(4): p. 340-357.

46. ISO, I. and T. IEC, *25051: Software engineering -- Software Product Quality Requirements and Evaluation (SQuaRE) -- Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing*. 2006, Geneva, Switzerland: International Organization for Standardization.

47. ISO, I., *IEC 25020 Software and System Engineering–Software Product Quality Requirements and Evaluation (SQuaRE)–Measurement Reference Model and Guide*, in *International Organization for Standarization*. 2007, Geneva, Switzerland: International Organization for Standardization.

48. ISO, I. and T. IEC, *25021: Software Engineering-Software Product Quality Requirements and Evaluation (SQuaRE)-Quality Measure Elements*. 2007, Geneva, Switzerland: International Organization for Standardization.

49. ISO, I., *IEC25010: 2011 Systems and Software Engineering–Systems and Software Quality Requirements and Evaluation (SQuaRE)–System and Software Quality Models*, in *International Organization for Standardization*. 2011, Geneva, Switzerland: International Organization for Standardization. p. 34.

50. ISO, I. and W. IEC, *25023*, in *System and Software Engineering–System and Software Product Quality Requirements and Evaluation (SQuaRE)–Measurement of System and Software Product Quality*. 2011, Geneva, Switzerland: International Organization for Standardization.

51. ISO, I., *25022*, in *System and Software Engineering–System and Software Product Quality Requirements and Evaluation (SQuaRE)–Measurement of Quality in Use*. 2012, Geneva, Switzerland: International Organization for Standardization.

52. Wagelaar, D., *D2. 1 Component-Based Frameworks.* 2004.

53. Girardi, M. and B. Ibrahim, *Using English to retrieve software.* Journal of Systems and Software, 1995. **30**(3): p. 249-270.

54. Lung, C.-H. and J.E. Urban. *An approach to the classification of domain models in support of analogical reuse*. in *ACM SIGSOFT Software Engineering Notes*. 1995. ACM. p. 169-178.

55. Podgurski, A. and L. Pierce, *Retrieving reusable software by sampling behavior.* ACM Transactions on Software Engineering and Methodology (TOSEM), 1993. **2**(3): p. 286-303.

56. Zaremski, A.M. and J.M. Wing, *Signature matching: A key to reuse*. Vol. 18. 1993: ACM.

57. Jeng, J.-J. and B.H. Cheng, *Specification matching for software reuse: A foundation.* ACM SIGSOFT Software Engineering Notes, 1995. **20**(SI): p. 97-105.

58. Prieto-Diaz, R. and P. Freeman, *Classifying software for reusability.* IEEE software, 1987. **4**(1): p. 6.

59. Dusink, L.M. and P.A. Hall, *Software Re-use, Utrecht 1989: Proceedings of the Software Re-use Workshop, 23–24 November 1989, Utrecht, The Netherlands*. 2013: Springer Science & Business Media.

60. Park, Y., *Software retrieval by samples using concept analysis.* Journal of Systems and Software, 2000. **54**(3): p. 179-183.

61. Dusink, L. and J. van Katwijk. *Reuse dimensions*. in *ACM SIGSOFT Software Engineering Notes*. 1995. ACM. p. 137-149.

62. Zaremski, A. and J. Wing, *Specification matching of software components.* ACM Transactions on Software Engineering and Methodology (TOSEM), 1997. **6**(4): p. 369.

63. Goguen, J., et al., *Software component search.* Journal of Systems Integration, 1996. **6**(1-2): p. 93-134.

64. Bowen, J.P. and M.G. Hinchey, *Ten commandments of formal methods.* Computer, 1995. **28**(4): p. 56-63.

65. Ben Khalifa, H., O. Khayati, and H. Ghezala. *A behavioral and structural components retrieval technique for software reuse*. in *Advanced Software Engineering and Its Applications, 2008. ASEA 2008*. 2008. IEEE. p. 134-137.

66. Inoue, K., et al., *Ranking significance of software components based on use relations.* IEEE Transactions on Software Engineering, 2005. **31**(3): p. 213-225.

67. Geisterfer, C.M. and S. Ghosh. *Software component specification: a study in perspective of component selection and reuse*. in *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006. Fifth International Conference on*. 2006. IEEE. p. 9 pp.

68. Fischer, B., *Specification-based browsing of software component libraries.* Automated Software Engineering, 2000. **7**(2): p. 179-200.

69. Lindig, C. *Concept-based component retrieval*. in *IJCAI95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*. 1995.

70. Qiang, J.X., *Organizing imperative programs for execution-based retrieval for reuse.* Electronic Theses and Dissertations, 2000.

71. Podgurski, A. and L. Pierce. *Behavior sampling: a technique for automated retrieval of reusable components*. 1992. ACM, Proceedings of the 14th international conference on Software engineering. p. 349-361.

72. Hall, R.J. *Generalized behavior-based retrieval*. 1993. IEEE Computer Society Press. p. 371-380.

73. Park, Y. and P. Bai, *Retrieving software components by execution.* Component-Based Software Engineering, 1998: p. 39.

74. Schumann, J. and B. Fischer, *NORA/HAMMR: Making deduction-based software component retrieval practical.* Proceedings of Automated Software Engineering (ASE-97), 1997: p. 246.

75. Hemer, D. *Specification matching of state-based modular components*. in *In Software Engineering Conference Tenth Asia-Pacific*. 2003. IEEE Computer Society. p. 446.

76. Penix, J. and P. Alexander, *Automated component retrieval and adaptation using formal specifications.* University of Cincinnati, Cincinnati, OH, 1998.

77. Zaremski, A.M. and J.M. Wing, *Signature matching: a tool for using software libraries.* ACM Transactions on Software Engineering and Methodology (TOSEM), 1995. **4**(2): p. 146-170.

78. Liddy, E.D., *Natural language processing.* Encyclopedia of Library and Information Science 2001.

79. Henninger, S., *Using iterative refinement to find reusable software.* Software, IEEE, 1994. **11**(5): p. 48-59.

80. Sugumaran, V. and V.C. Storey, *A semantic-based approach to component retrieval.* ACM SIGMIS Database, 2003. **34**(3): p. 8-24.

81. Yao, H. and L. Etzkorn. *Towards a semantic-based approach for software reusable component classification and retrieval*. 2004. ACM,Proceedings of the 42nd annual Southeast regional conference. p. 115.

82. Antoniol, G., et al. *Information retrieval models for recovering traceability links between code and documentation*. in *Software Maintenance, 2000. Proceedings. International Conference on*. 2000. IEEE. p. 40-49.

83. Järvelin, K. and J. Kekäläinen, *Cumulated gain-based evaluation of IR techniques.* ACM Transactions on Information Systems (TOIS), 2002. **20**(4): p. 422-446.

84. Salton, G. and C. Buckley, *Improving retrieval performance by relevance feedback.* Readings in information retrieval, 1997. **24**(5).

85. Breitman, K.K., and Julio Cesar Sampaio do Prado Leite. *Ontology as a requirements engineering product*. in *Requirements Engineering Conference,*. 2003. 11th IEEE International. p. 309-319.

86. Nuseibeh, B. and S. Easterbrook. *Requirements engineering: a roadmap*. in *Proceedings of the Conference on the Future of Software Engineering*. 2000. ACM. p. 35-46.

87. Castañeda, V., et al., *The use of ontologies in requirements engineering.* Global Journal of Research In Engineering, 2010. **10**(6).

88. Ruiz, F. and J.R. Hilera, *Using ontologies in software engineering and technology*, in *Ontologies for software engineering and software technology*. 2006, Springer. p. 49-102.

89. Kalfoglou, Y. and M. Schorlemmer, *Ontology mapping: the state of the art.* The knowledge engineering review, 2003. **18**(01): p. 1-31.

90. Fürber, C., *Semantic Technologies*, in *Data Quality Management with Semantic Technologies*. 2016, Springer. p. 56-68.

91. McGuinness, D.L. and F. Van Harmelen, *OWL Web Ontology Language Overview. W3C Recommendation.* Latest version is available at http://www. w3c. org/TR/owl-features, 2004. **10**(10): p. 2004.

92. Davis, R., H. Shrobe, and P. Szolovits, *What is a knowledge representation?* AI magazine, 1993. **14**(1): p. 17.

93. Gu, T., H.K. Pung, and D.Q. Zhang, *A service-oriented middleware for building context-aware services.* Journal of Network and computer applications, 2005. **28**(1): p. 1-18.

94. Dzung, D.V. and A. Ohnishi. *A verification method of elicited software requirements using requirements ontology*. in *2012 19th Asia-Pacific Software Engineering Conference*. 2012. p. 553-558.

95. Al Balushi, T.H., et al. *ElicitO: a quality ontology-guided NFR elicitation tool*. in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. 2007. Springer: Berlin, Heidelberg. p. 306-319.

96. Kassab, M., O. Ormandjieva, and M. Daneva. *An Ontology based approach to non-functional requirements conceptualization*. in *ICSEA'09. Fourth International Conference on Software Engineering Advances*. 2009. IEEE. p. 299-308.

97. Jung, H.-W., S.-G. Kim, and C.-S. Chung, *Measuring software product quality: A survey of ISO/IEC 9126.* IEEE software, 2004. **21**(5): p. 88-92.

98. Borst, W.N., *Construction of engineering ontologies for knowledge sharing and reuse*. 1997: University Twente.

99. Dobson, G., S. Hall, and G. Kotonya. *A domain-independent ontology for non-functional requirements*. in *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*. 2007. IEEE. p. 563-566.

100. Lopez, C., L.M. Cysneiros, and H. Astudillo. *NDR ontology: Sharing and reusing NFR and design rationale knowledge*. in *Managing Requirements Knowledge, 2008. MARK'08. First International Workshop on*. 2008. IEEE. p. 1-10.

101. Wang, X., F. Fang, and L. Fan. *Ontology-Based Description of Learning Object*. in *International Conference on Web-Based Learning*. 2008. Springer. p. 468-476.

102. IEC, I., *9126-1 (2001). Software Engineering Product Quality-Part 1: Quality Model.* International Organization for Standardization, 2001.

103. Grau, G., et al. *DesCOTS: a software system for selecting COTS components*. in *Euromicro Conference, 2004. Proceedings. 30th*. 2004. IEEE. p. 118-126.

104. Bhaumik, S.S. and R. Rajagopalan, *Elicitation techniques to overcome knowledge extraction challenges in 'as-is' process modelling: perspectives and practices.* International Journal of Process Management and Benchmarking, 2009. **3**(1): p. 47-59.

105. Christensen, E., et al., *Web services description language (WSDL) 1.1*. 2001.

106. Guha, R., R. McCool, and E. Miller. *Semantic search*. in *Proceedings of the 12th international conference on World Wide Web*. 2003. ACM. p. 700-709.

107. Li, C., R. Pooley, and X. Liu, *Ontology-Based Quality Attributes Prediction In Component-Based Development.* International Journal of Computer Science & Information Technology (IJCSIT), 2010. **2**(5): p. 12-29.

108. Newell, D., et al. *Models for an Intelligent Context-Aware Blended m-Learning System*. in *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*. 2015. IEEE. p. 405-410.

109. Mohamed, A., G. Ruhe, and A. Eberlein. *COTS selection: past, present, and future*. in *Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the*. 2007. IEEE. p. 103-114.

110. Mens, T., *Introduction and roadmap: History and challenges of software evolution*, in *Software evolution*. 2008, Springer. p. 1-11.

111. Pérez, F. and P. Valderas. *Allowing end-users to actively participate within the elicitation of pervasive system requirements through immediate visualization*. in *Requirements Engineering Visualization (REV), 2009 Fourth International Workshop on*. 2009. IEEE. p. 31-40.

112. Wilson, W.M., L.H. Rosenberg, and L.E. Hyatt. *Automated analysis of requirement specifications*. in *Proceedings of the 19th international conference on Software engineering*. 1997. ACM. p. 161-171.

113. Berry, D.M. and E. Kamsties, *Ambiguity in requirements specification*, in *Perspectives on software requirements*. 2004, Springer. p. 7-44.

114. Subramanyam, R. and M.S. Krishnan, *Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects.* Software Engineering, IEEE Transactions on, 2003. **29**(4): p. 297-310.

115. Osborne, M. and C. MacNish. *Processing natural language software requirement specifications*. in *Proceedings of the Second International Conference on Requirements Engineering*. 1996. IEEE. p. 229-236.

116. Glinz, M. *On non-functional requirements*. in *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*. 2007. IEEE. p. 21-26.

117. Mairiza, D., D. Zowghi, and N. Nurmuliani. *An investigation into the notion of non-functional requirements*. in *Proceedings of the 2010 ACM Symposium on Applied Computing*. 2010. ACM. p. 311-317.

118. Lassila, O. and R.R. Swick, *Resource description framework (RDF) model and syntax specification.* 1999.

119. Decker, S., et al., *The semantic web: The roles of XML and RDF.* IEEE Internet computing, 2000. **4**(5): p. 63-73.

120. Prud, E. and A. Seaborne, *SPARQL query language for RDF.* W3C recommendation, 2006.

121. Fernández, M., et al., *Semantically enhanced information retrieval: An ontology-based approach.* Web semantics: Science, services and agents on the world wide web, 2011. **9**(4): p. 434-452.

122. Moraes, R., et al., *Component-based software certification based on experimental risk assessment*, in *Dependable Computing*. 2007, Springer. p. 179-197.

123. Dickinson, I., *Jena ontology api.* On the WWW, at http://jena. sourceforge. net/ontology/index. html [accessed 10/12/2015], 2009.

124. Seaborne, A., *ARQ-A SPARQL Processor for Jena.* Obtained through the Internet: http://jena. sourceforge. net/ARQ/, [accessed 10/12/2015], 2010.

125. Booch, G., *The unified modeling language user guide*. 2005: Pearson Education India.

126. Inc, B., *Systems Development Lifecycle: Objectives and Requirements.* New York: Queensbury, 2003.

127. Math, C., *The apache commons mathematics library.* Np, nd Web, 2016. **9**.

128. Otto, M. and J. Thornton, *Bootstrap.* Twitter Bootstrap, 2013.

129. Hansson, D.H., *Ruby on rails.* Website. Projektseite: http://www. rubyonrails. org, 2009.

130. Hickson, I. and D. Hyatt, *HTML5: A vocabulary and associated APIs for HTML and XHTML.* W3C Working Draft edition, 2011.

131. Overson, J. and J. Strimpel, *Developing Web Components: UI from jQuery to Polymer*. 2015: " O'Reilly Media, Inc.".

Appendix

The following form has been shared with the experiment participants.

Experiment Questions

Component Selection Tool

* Required

What do you understand NFRs to be? *

Do you think values are necessary when specifying NFRs or is simply selection of NFRs sufficient ? *

- Values
- NFRs
- Other:

Overall, describe your experiences in using this tool? *

What difficulties did you have when using this tool ? *

What benefits do you expect to see from using a tool like this? *

When you think about this new tool, do you think of it as something developers might NEED or as something developers might WANT? *

- NEED
- WANT
- Other:

Please explain your reason for the answer to previous question. *

What are the things that you like most about this tool? *

If we had to build this tool again, what would you change in terms of processes ? *

What do you understand NFR-based Component descriptions to be? *

What type of component description do you prefer to use ? *

- Functional Requirement base
- Non Functional Requirement base
- Other:

Please explain your reason for the answer to previous question. *

For each scenario :

How long did it take to analyse the scenario? *

How close are the scenarios to real world ones? *

What difficulties did you have in analysing the scenario? *

Education: What is the highest degree or level of school you have completed?

Education: What is your major (specialisation)?

Work: How many years' work experience do you have?

Powered by Google