

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/260835083>

A Tactic-Based Framework to Evaluate the Relationships between the Software Product Quality Attributes

Article · May 2012

CITATIONS

7

READS

248

3 authors:



Saleh Aldaajeh

Institute of Applied Technology

6 PUBLICATIONS 32 CITATIONS

SEE PROFILE



Rafa E. Al-Qutaish

61 PUBLICATIONS 417 CITATIONS

SEE PROFILE



Fuad El-Qireem

Al-Zaytoonah University of Jordan

7 PUBLICATIONS 16 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



An Educational and Diagnostic Tool to Help Autism Spectrum Disorder Children [View project](#)



Business Continuity Planning in Divisional / Functional Authority Organisation Context [View project](#)

A Tactic-Based Framework to Evaluate the Relationships between the Software Product Quality Attributes

Saleh H. Al-Daajeh⁽¹⁾, Rafa E. Al-Qutaish⁽²⁾, and Fuad Al-Qireem⁽¹⁾

(1) Al-Zaytoonah University of Jordan, Airport Street, Amman (Jordan)
E-mail: {saleh.aldajah, alqerim}@yahoo.ca

(2) Al Ain University of Science & Technology - Abu Dhabi Campus, Abu Dhabi (UAE)
E-mail: rafa@ieee.org

ABSTRACT

Understanding quality attributes relationships impersonates a vital role in sustaining a sufficient level of quality to a software system. Furthermore, calibrating the relationships between system's quality attributes will not only affect its quality but also may reduce the development cost. However, this paper introduces a framework to explore the nature of the relationships between different quality attributes at a low level with the aid of architectural strategies (i.e. tactics). Additionally, this framework enables the software architects to calibrate and adjust the quality attributes relationships using two pieces of information, that is, tactics and scenarios. Furthermore, the developed framework enables software architects to have a rough measurement of the strength of the relationships between the quality attributes.

Keywords: ISO 9126, Quality Attributes Relationships, Quality Characteristics, Quality Models, Scenarios, Software Architectural Strategies, Software Product Quality Attributes, Tactics.

1- INTRODUCTION

Managing software quality is a key characteristic in its own, or in its implications. However, achieving quality to a software system is never isolated to a certain technique or a distinct stage. The ultimate goal of software development is to introduce quality to a software system in a cost effective way, that is, within short time and with less cost. Thus, it is necessary to consolidate the achievement of quality to a software system at the earliest stages of its development.

The software quality life cycle starts with determining the non-functional requirements which describe how the system should respond to a certain stimulus and how the system is proposed to perform things [28]. However, the software quality is defined as the degree to which a system, component or process meets specified requirements, customer or user needs and expectations [14]. When developing a software system, it is necessary to sustain a sufficient level of quality to the software product and its development process

[27]. Thus, introducing quality to software at earlier stages than doing it at later ones [25, 28].

McCall [20] stated that the relationships between quality attributes exist. Furthermore, the understanding of the quality attributes relationships minimize the chances of the occurrence of unwelcome system behavior during runtime, and it increases the chance that the developed system will meet its specifications by selecting the appropriate design decisions at early stages of the systems development process [21]. Moreover, the benefits of understanding the nature of quality attributes' relationships will aid in deciding which quality attributes to prioritize and which one to forsake during the development stage [27]. Unfortunately, up to present the quality attributes relationships were investigated from a higher level only and not from a lower level. Furthermore, the process of adjusting the relationships between the different quality attributes of a software system is yet not discussed. Nevertheless, this paper propose a framework to enable software architects to investigate the nature of the relationships between different quality attributes at a lower level (solutions level), and provides indicators which can help the architects to adjust the relationships between system quality attributes to consolidate the achievement of the overall quality for the system.

The paper is organized as follows: Section 2 presents a background and the motivation for this research paper, and Section 3 illustrates the intended framework. Section 4 shows the framework execution, while Section 4 analyses the framework. Section 5 discusses and validates the framework, and finally, Section 6 concludes the paper.

2- BACKGROUND AND MOTIVATION

This study is primarily concerned with exploring the nature of the relationships between the quality attributes at the software architecture stage. However, software quality is one of the thematic blocks in which this study is focused on. Software quality is further explained in subsection 2.1. The software architecture plays a vital role in achieving software quality attributes [11] and it is discussed in subsections 2.2 and 2.3. The development of the software system may have several options which require trade-offs; also it is necessary to compromise between different quality attributes to sustain a sufficient level of quality to a software system [5]. The trade-offs are further discussed in subsection 2.4. Finally, the recent studies which investigate the relationships between quality attributes are depicted in subsection 2.5.

2-1 Software Quality

The achievement of quality for a software system is not a clean-up process for errors and defects [2]. Therefore, techniques such as inspection, reviews, and testing have already been used in practice since decades as candidate techniques to introduce quality to software [2, 10]. Nevertheless, the achievement of quality for software systems is usually done via three basic ways; the oldest two methods are undertaken nowadays as validation and verification activi-

ties, while the third and the recent method is engineering quality for software systems [10]. These methods are usually used together for interchanging goals to sustain a sufficient level of quality to software systems throughout their development processes [14]. Additionally, it is widely accepted that engineering quality for software systems is less expensive, especially when introducing quality to software at earlier stages than doing it at later ones [10, 25, 28]. For example, introducing quality to software at the software architecture stage costs less than introducing it during at the maintenance stage. However, engineering quality for software is the recent approach adopted to achieve quality for software systems. This method requires a large effort in coordinating activities and selecting the appropriate tools and engineering techniques.

Software system quality achievement's life cycle is continuous throughout the software development process where the aforementioned methodologies / approaches are adopted in almost in every stage of the software development processes. Also, the software quality achievement's life cycle is never isolated to a distinct development process of the product life cycle or to a specific methodology. Thus, the afore stated methodologies are continuing and overlapping over all development processes with an interchanging between these methods goals and practices, and to be undertaken for the different trade-offs embracing different factors which may have an influence on the system's quality [7, 10]. Furthermore, software quality achievement is strongly related to the achievement level of the system's predefined attributes.

In software engineering literature, there are several quality models which are used by researchers and practitioners. However, Al-Qutaish [1] has compared and analyzed five quality models and concluded that the ISO 9126 quality model is the most useful and used one. Thus, as a case study, the relationships between the maintainability and safety and each of the six ISO 9126 [16] internal and external quality characteristics will be assessed in section 4. Furthermore, this paper proposes a framework to assess the relationships between quality attributes of any quality model.

2-2 Engineering Quality via Software Architecture

In most cases the second phase of a software development life cycle is the high-level design phase or the software architecture stage [23]. However, software architecture in terms of components and connectors supports the achievement of the system's quality attributes by providing design decisions and specifications that satisfy the means of quality attributes. The IEEE provides the IEEE Std. 1471 standard [15] (it is adapted by ISO as ISO/IEC 42010 standard [17]) that defines the software architecture as the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution. However, the software architecture society has been active in this field and it has shown how software architecture constrains the achievement of quality [4, 5, 7, 25, 31]. Furthermore, Bass et al. [6] has defined the software architecture as the structure or structures of the system which compro-

mise software components, the externally visible properties of those components, and the relationships between them. In addition, software architecture is documented via multiple methodologies and structured via different styles. Additionally, a study of monitoring the progress of achieving quality throughout the software life cycle stages indicates that the software architecture stage constraints up to 45% of the overall achievement of the systems' quality attributes [11].

The achievement of quality attributes for a system in software architecture can be separated into two levels, that is, requirements and solutions. The software architecture community has developed several frameworks used to elicit, specify, and categorize the quality requirements [6, 18, 19]. These frameworks are used to create what is known as quality attributes general scenarios which help in developing quality attributes concrete scenarios, and therefore assist in evaluating the software architectures [6]. Thus, such general scenarios can be used as a template to define and construct multiple concrete scenarios for quality attributes depending on how many the quality requirement embraces of the system's non-functional requirements [12]. Quality attributes concrete scenarios are methods to create a description of the problem (constraint or requirement) which ask whether the software architecture candidate solutions satisfy the desired non-functional requirements or not [12].

The solution for achieving software quality attributes during the software architecture stage is supported by selecting the appropriate architectural strategies (or tactics) [4, 6]. However, tactics are known as design decisions that influence and control the response of quality attributes [4, 6]. There are several tactics suggested to support the achievement of different quality attributes [6, 30]. Nevertheless, early design decisions adopted in different design patterns have a considerable impact on the achievement of various quality attributes of a system. Furthermore, Bosch et al. [9] has stated that there is a lack of details and knowledge on how the design strategies influence the systems' quality attributes. Thus, it pays off to assess the design decisions impact on systems' quality attributes relationships at this stage.

2-3 Software Architecture Evaluation

To achieve quality for a software system at the architecture stage, it is important to evaluate the system architecture with respect to the desired quality and the implemented system requirements. In general, the evaluation of software architecture aims at providing evidences to assure whether the architecture is suitable with respect to its functional and non-functional requirements or not.

Moreover, software architecture evaluation methods are usually based on critical analysis of how the system fulfils its specification. However, there are several methods that can be used to assess software architecture and predict the quality of software at the architectural level (e.g. ATAM, SAAM, etc.) [6].

Kazman et al. [18] has classified the software architecture evaluation methods into two general categories, that is, qualitative analysis and quantitative measurement. The qualitative analysis of software architecture is usually based on questionnaires, checklists, and/or scenarios. On the other hand, quantitative measurements consist of simulations and metrics, for example, modeling and testing. Furthermore, quantitative measurement aims at estimating the fulfillment of a quality attribute in terms of probabilities [18, 21]. Nevertheless, there are several studies conducted to evaluate architectural styles / patterns that support different quality attributes [6, 8, 26]. Berander et al. [7] and Svahnberg et al. have introduced further recommendations on software architecture evaluation methods.

Software architecture evaluation does not only help the evaluators to be able to reason about the achievement of software quality attributes and to have a certain level of predictability of the systems behavior, but it also helps to select suitable tactics to achieve the desired quality attributes, and therefore supports the trade-offs and in turn the selection of an appropriate architectural pattern.

Objective reasoning is one of the approaches used to assess the achievement of the software quality requirements in software architecture through reasoning-based logical arguments [5]. However, since design strategies are experience-based, thus the evaluation method adopted in this research is objective reasoning.

2-4 Trade-offs

Trade-offs are subconsciously based on a daily basis. However, system quality attributes are among the important subjects that in some situations require trade-offs especially when facing quality requirements conflicts. Furthermore, trade-offs differ from one situation to another, and thus a rigorous analysis might be required. Any technique that helps to make or consolidate the trade-off process or decision can be considered as a trade-off technique.

The trade-off process can be investigated from two dimensions, that is, the used techniques, and the depth of the analysis of the trade-offs processes. However, Berander et al. [7] stated that there are several categories for trade-offs techniques such as experienced based, model based, and mathematically based. Furthermore, the analysis of the situation where the trade-off process is required aims to appraise the possible factors which may affect the choice of the solution. Therefore, the critical part of trade-offs processes is to define the factors. Moreover, trade-offs can be made at different levels and can also be constructed of several sub-trade-offs within one level which makes trade-offs dependent on the depth and strength of the required analysis. Software architects are the ones who make the final decisions on how the system shall meet its specifications and customers expectations. Different types of software systems trade-offs are performed at two main levels with respect to their quality. The first level is to prioritize the desired system quality attributes, for exam-

ple, a landing system must have high availability while it is not required to be reliable for longer periods. The second level of trade-offs is the selection of a software architecture pattern / style that supports the pre-prioritized quality attributes to the best possible way [7].

Furthermore, in the second level of trade-offs there is a set of sub-trade-offs which is needed in the form of technical reviews. However, the intention of technical reviews is to evaluate the solution(s) against the predefined specifications and compliance with the standards and other documentations [22]. In such sub-trade-offs, software architects are required to select and calibrate the best candidate tactics that consolidate the relationship between the system's quality attributes by supporting the achievement of other quality attributes or less hindering it. Therefore, software architects must take into considerations the nature of the interrelationships between system's quality attributes in order to select and calibrate the best representative architectural strategies, and thus he needs to create an architecture that meets the predefined specifications on the desired quality of the potential developed systems.

2-5 Related Work

As stated in the introduction, the understanding of the quality attributes relationships is important to sustain a sufficient level of quality to a system and its development processes [21, 27]. However, the very recent studies that investigate the relationships between quality attributes are based on different approaches. For example, the studies conducted by Henningsson et al. [13] and Zulzalil et al. [32] used the experience-based approach. These studies are interpreting the relationships between quality attributes throughout the system development and usage. Furthermore, Svahnberg and Henningsson [27] have investigated the relationships between the quality attributes by combining different views, such as academic, industry, and literature; also they concluded a match between these views. Moreover, these studies investigate quality attributes relationships from a higher-level and do not consider a specific development stage [27, 32]. Table 1 provides a match between the aforementioned studies' results [27, 32].

Table 1 Quality attributes relationships [3, 29].

	Functionality	Efficiency	Reliability	Usability	Maintainability
Efficiency	-				
Reliability	+	0			
Usability	+	-	+		
Maintainability	+	-	+	0	
Portability	0	-	0	0	+
- negative relationship + positive relationship 0 no relationship					

3- QUALITY ATTRIBUTES RELATIONSHIPS FRAMEWORK

The Zhu et al. [31] model has been used in the design of this framework. Furthermore, this model describes the skeleton of the relationship between patterns, tactics, evaluation, and pattern validation. However, our proposed framework for assessing the relationships between quality attributes consists of three main stages, that is, preparation, scenario development, and evaluation stages. Figure 1 depicts the structure of this framework.

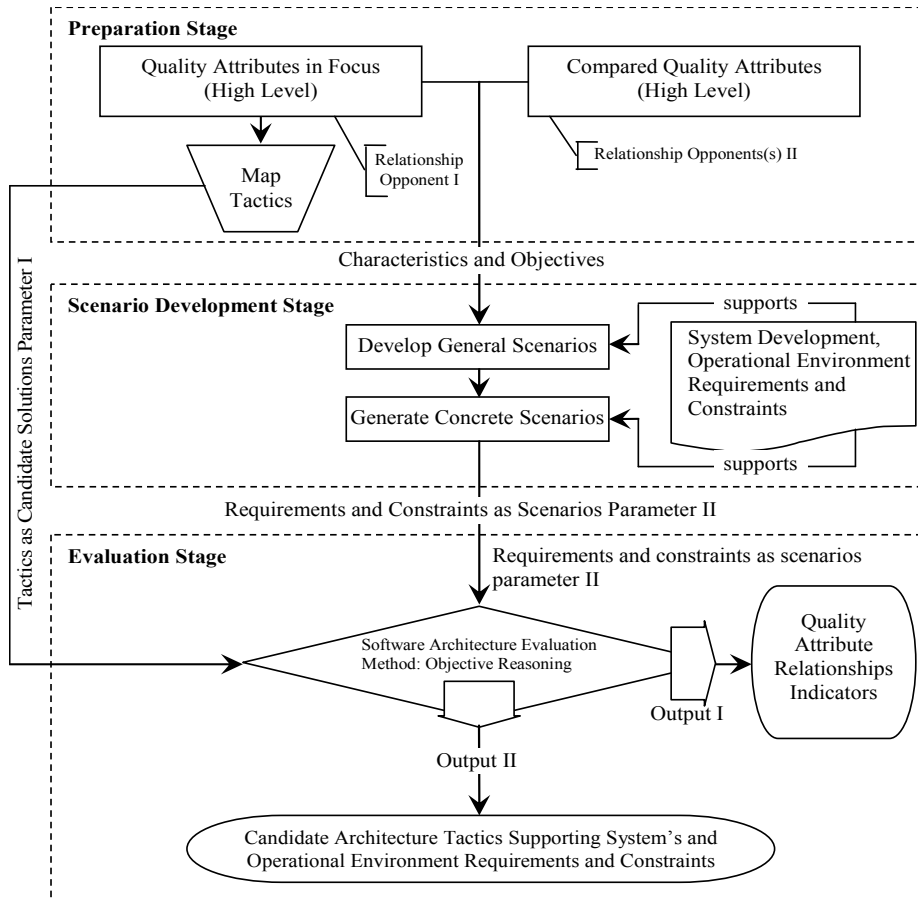


Figure 1 Quality attributes relationships reasoning framework.

3-1 Preparation Stage

This stage consists of the following two main steps:

1. Identifying the relationships opponents as the 'quality attribute in focus' and

the 'compared quality attribute(s)'. The inputs of this step are firstly the relationship opponents, that is, the Maintainability and Safety quality attributes as the 'quality attributes in focus' and being the first opponent in the relationship. Secondly, the 'compared quality attribute(s)' are the dependability and other quality attributes found in the ISO 9126 [16] quality model (i.e. functionality, efficiency, usability, portability, reliability, and safety).

2. Mapping architectural strategies (tactics) as candidate solutions to achieve the 'quality attribute in focus'. This step is concerned with identifying candidate solutions in terms of tactics to the 'quality attribute in focus'. In addition, this step forwards the first independent parameter needed for the Evaluation stage.

3-2 Scenario Development Stage

Since quality requirements are seldom elicited and recorded in a disciplined way [6]. Therefore, there is more than a single matrix that aims at capturing the important elements of quality requirements [31]. However, Bass et al. [6] proposed a matrix which we considered in this paper, and it is further described in table 2. However, this stage consists of two successive steps:

1. Developing quality attributes general scenarios: This step concerned with developing quality attributes general scenarios. These scenarios aim to capture the relationship opponents 'quality attribute in focus' and the important elements, concerns, and the development and operational constraints of the 'compared quality attributes'.
2. Constructing quality attributes concrete scenarios: This step focused on describing the opponents' important elements with respect to the systems' requirements. The main objective of this step is to fully address the studied quality attributes. However, quality attributes concrete scenarios can be supported by additional data such as system technical information, requirements, and the development and operational constraints. Furthermore, this step forwards the second independent parameter needed for the evaluation stage.

Table 2 Quality attributes scenarios' matrix illustrated from [17].

Elements	Description
Source of Stimulus	An entity (human, computer, etc.) that generate the stimulus.
Stimulus	A condition that need to be considered when it arrives at a system.
Simulated artifacts	Some artifacts that are simulated (e.g. whole system, parts of the system).
Environment	A system's condition when a stimulus occurs.
Response	The activity undertaken after stimulus arrival.
Response Measure	The response to the stimulus should be measurable in some fashion so that the requirement can be tested.

3-3 Evaluation Stage

In this step, the aforementioned objective reasoning method is adopted to evaluate the impact of candidate tactics of 'quality attributes in focus' on the 'compared quality attribute(s)'. However, this evaluation method has two main parameters that are produced in the preparation and scenario development stages. In addition, different factors that may influence the assessment of the tactic impact are considered, such as design rational, domain context, and implementation factors.

The quality attributes relationships are given a minus sign (-) or letter N when the relationship is negative between the two parts of the relationship 'quality attribute in focus' and the 'compared quality attributes'. Furthermore, the negative relationship indicates that the achievement of the quality attribute in focus will hinder the given compared quality attribute(s). The positive (or supportive) relationships are presented using the plus sign (+) or letter P. The positive relationship means that the 'quality attribute in focus' will support the achievement of the 'compared quality attribute(s)'.

The evaluation of the relationship strength and severity is measured based on an ordinal scale, that is, from 0 to 4, where the zero value or letter I represent a no impact and thus an independent relationship, and +4 or -4 values represents a very strong positive impact supporting or a severe impact hindering the achievement of the compared quality attribute(s). Since the scale used in the evaluation was placed on an ordinal scale, then the median value between the results' deviation represents the strength or severity of the relationship between the two parts of the relationship. The final representation of the relationship between the quality attributes is based on the evaluation of the adopted solutions' impact on other quality attributes. For example, if most of the candidate solutions have a negative impact on achieving a given quality attribute, then the relationship between these quality attributes is negative. Furthermore, the evaluation of the relationship nature that is based on candidate tactics impact is described in following Equation:

$$R(a,b) = \sum_{i=1}^n E(a_i) \quad (1)$$


Where, a is the quality attribute in focus, b is the compared quality attribute, R is the relationship function between a and b , n is the number of candidate tactics i to satisfy the achievement of a , and E is the evaluated tactics of a .

4- FRAMEWORK EXECUTION

In this section, the framework is applied to explore the nature of the relationship between the maintainability and safety quality attributes as 'quality

attributes in focus' and the functionality, efficiency, usability, portability, reliability, and maintainability ISO 9126 [16] quality attributes/characteristics as 'compared quality attributes'.

Although the safety quality attribute is not enclosed in the ISO 9126 [16] quality model, Avizienis et al. [3] has defined the safety quality attribute as the non-occurrence of catastrophic consequences for the users and in the operation environment. However, this quality attribute describes how the system should deal with mishaps and/or failures when they occur. Moreover, the safety quality attribute is undertaken in the execution of the quality attributes relationships reasoning framework to explore the nature of the relationships between quality attributes. Also, how the maintainability quality attribute relationship interchanging with the other relationship opponent. Since the relationship opponents have been identified, the next step in the preparation stage is to identify the candidate architectural strategies (tactics) that satisfy the achievement of the 'quality attribute in focus'.



The process of identifying tactics as the candidate solutions to a given quality attributes is dependent on quality attributes' concerns and characteristics. For example, the maintainability quality attribute encloses multiple sub-characteristics and concerns [3, 7]. Therefore, the identification of the candidate tactics to achieve the maintainability quality attribute is based on the achievement of its concerns and sub-characteristics.

The candidate tactics for the safety quality attribute are illustrated directly from the study performed by Wu and Kelly [30]. However, the maintainability and safety quality attributes candidate tactics are illustrated in Table 3, and elicited from Bass et al. [6] and Wu and Kelly [30] researches. In addition, there are several proposed tactics to achieve different quality attributes [6].

The second stage of this framework is to develop general and concrete scenarios for both 'quality attributes in Focus' and the 'compared quality attribute(s)'. The significant difference between the general and concrete scenarios are that the general scenarios are the templates by which enables the software architect to generate a concrete scenario depicting necessary elements necessary to achieve a given quality attribute. However, concrete scenarios are usually supported with more detailed and low level information. Therefore, in order to consolidate the generation of quality attributes concrete scenarios, thus the quality attributes' scenarios developed in this research are supported by the technical information provided by the Swedish Space Cooperation (SSC) for the SMART-1satellite system [24]. Smart-1 satellite system represents Embedded System software in which a degradation in these types of systems' quality may result in catastrophic consequences. Smart-1 satellite system operational environment and performed functions are more or less can be illustrated in other types of software systems such as computer-based software systems, web-based software systems, information systems, etc. Therefore, the use of the Smart-1 satellite system as an object to execute the developed framework will give the opportunity to implement this framework on

other types of software system and eliminate the validity threat of generalizing the results for other types of software systems [29].

From implementation point of view, quality attributes' general and concrete scenarios can be further distinguished from the number of the non functional requirements described in a given scenario. Quality attributes' general scenarios may embrace different non functional requirements. In addition, quality attributes' concrete scenarios are more concise by embracing a certain requirement which influences a given quality attribute of a software system.

However, the following is an example of a general scenario depicting the important elements of the maintainability quality attribute for the SMART-1 satellite system:

The satellite system proposed to operate for a long period of time (Environment). During this long life, the satellite system is most likely required to be upgraded (Stimulus) from control base maintainers (Source of Stimulus). The system (Artifact) shall provide the opportunity to be upgraded remotely (Response) and shall not exceed the duration of maximum one operation week to maintainer to implement upgraded features or functionalities (Response Measure).

Table 3 Candidate architectural tactics for the maintainability and safety quality attributes.

Safety	Maintainability
Failure Detection:	Manage Input / Output:
	Record / Playback
Timeout	Separate Interfaces From Implementation
Time Strap	Specialized Access Routines /Interfaces
Sanity Checking	Localize Changes :
	Semantic Coherence
Failure Containment:	Anticipate/ Expected Changes
	Generalize Module
Redundancy	Limit Possible Options
Replication	Abstract Common Services
Functional Redundancy	Prevention of Ripple Effect:
Analytical Redundancy	Hide Information
	Maintain Existing Interface
Recovery:	Restrict Communication Paths
	Use an Intermediary
Fix the Error	Internal Monitoring:
Rollback	Built in Monitors
Degradation	Defer Binding Time:
Configuration	Runtime Registration
	Configuration Files
Masking:	Polymorphism
	Component Replacement
Voting	Adherence to Define Protocols

Moreover, Table 4 below shows the general scenario for maintainability quality attribute.

Table 4 Maintainability general scenario.

Elements	Description
Source of Stimulus	Satellite control base.
Stimulus	Software and Hardware Upgrades.
Simulated artifacts	Satellite System components.
Environment	Normal Condition
Response	Ability to upgrade remotely (on-line).
Response Measure	Duration of achieving necessary upgrades shall not exceed the duration of one week.

The execution of the developed framework intends to address the relationships between the high level quality attributes as the relationship opponents. However, since the evaluation method is based on experiences and logical argumentation, then a survey was distributed among different participants that are carefully selected based on the stratified sampling methodology from different backgrounds related to the scope of this research. Furthermore, the research survey was a take home survey for which the participants were given a sufficient time to answer the survey questions and be able to use the supplementary data to reasonably answer the research questions. The approach to collect answered survey was done via email. Table 5 depicts the surveys' details. In addition, results illustrated from the study survey are depicted in Tables 6 and 7.

Table 5 The survey details.

Participants Role	No.	Quality Attributes Assessed
MSc. Software Engineering students	30	Maintainability, and Safety
Systems Security Engineers	8	Safety
Academia	17	Maintainability, and Safety
Software Designers	8	Maintainability, and Safety
System Developers	20	Maintainability
System Testers	5	Maintainability
Total No. of Participants	88	Assessing quality attributes relationships based on tactics' impact

Table 6 Safety candidate impact evaluation.

Candidate Tactics	Safety Impact Evaluation on						
	Functionality	Efficiency	Usability	Portability	Reliability	Maintainability	Level of Agreement
Failure Detection							
Time out	0	+4	+2	-1	0	0	60%
Time Strap	0	-1	0	+4	+4	+4	60%
Sanity Checking	0	-1	0	+4	+4	+4	80%
Failure Containment							
Redundancy	+4	-2	0	+2	+4	+4	80%
Replication	+4	-2	0	+2	+2	+4	80%
Functional Redundancy	+2	-2	0	+2	+2	+2	60%
Analytical Redundancy	0	-2	0	0	0	0	80%
Recovery Re-Introduction							
Fix the Error	+2	-2	0	0	+2	+2	80%
Rollback	0	-4	0	0	+3	+2	60%
Degradation	-4	0	0	0	-3	0	80%
Reconfiguration	+4	-1	0	+1	0	+4	60%
Masking							
Voting	-1	-4	0	0	+4	-2	100%
Relationships	P	N	P	P	P	P	
Impact Median Value	0	0	+1	+1.5	+0.5	+1	

5- ANALYSIS

This research is designed based on two main objectives, that is, the first one is to understand the nature of the inter-relationship between the different quality attributes of systems based on the architectural solution provided by the software architecture community, and the second objective is to increase the quality for a software system at early stages of its development by evaluating the impact of using tactics.

The results of this research paper indicate that the relationships between quality attributes are interchangeable. For example, when achieving the maintainability quality attribute by using the proposed architectural tactics, then a slight support on achieving the reliability and safety quality attributes is noted and therefore creates a positive relationship. However, when focusing on achieving the safety quality attribute, the use of the proposed architectural tactics has a minor impact on supporting the achievement of the maintainability quality attribute and therefore creates another positive relationship with less magnitude.

Table 7 Maintainability candidate impact evaluation.

Candidate Tactics	Maintainability Impact Evaluation on						
	Functionality	Efficiency	Usability	Portability	Reliability	Safety	Level of Agreement
Manage Input / Output							
Record/Playback	0	0	0	0	+1	+2	80%
Separate Interfaces from Imp.	0	0	0	+4	+2	+2	100%
Specialized Access Routines/Interfaces	0	+4	0	-1	0	0	60%
Localize Changes							
Semantic Coherence	0	-2	0	+4	0	0	80%
Anticipate Changes	0	-2	0	+4	0	0	80%
Generalize Module	0	-1	0	+2	0	0	100%
Limit Possible Options	-2	+1	-2	0	0	+1	60%
Abstract Common Services	0	-2	0	+4	0	0	80%
Prevention of Ripple Effect							
Hide Information	0	-2	0	+4	0	0	100%
Maintain Existing Information	0	-1	0	+2	0	0	80%
Restrict Communication Paths	0	+2	0	+1	0	0	60%
Use an Intermediary	0	-4	+2	+2	+4	+3	80%
Internal Monitoring							
Built-in Monitors	0	-1	+2	-3	+4	+4	100%
Defer Binding Time							
Run-Time Registration	0	-4	0	+2	0	0	80%
Configuration Files	+2	0	+2	0	0	0	100%
Polymorphism	0	-2	0	0	0	0	80%
Component Replacement	+4	-2	+2	+2	0	0	100%
Adherence to Define Protocols	+2	+2	0	+3	0	0	60%
Relationships							
	P	N	P	P	P	P	
Impact Median Value							
	+1	0	0	+0.5	+2	+2	

There are many tactics that can be used to satisfy a certain concern of a quality attribute [6]. Such tactics have different levels of impact on the achievement of software systems quality attributes. Thus, the software architects have the option to select and calibrate the appropriate architectural solutions or tactics to satisfy a certain concern for a quality attribute and therefore enable them to balance the relationship between the software system quality attributes to meet certain system specifications. For example, to achieve the safety quality attribute and satisfy the 'failure Containment' as one of its concerns, software architects are recommended to select the redundancy tactic. The impact of using such a tactic will deter less the achievement of other quality attributes of a software system. On the other hand, the selection of the analytical redundancy architectural tactic constrains the achievement of the efficiency quality attribute for the software system. Moreover, the median value of impact level of proposed architectural tactics represents the strength of the relationship between the quality attributes of a software system. For example, when utilizing the proposed tactics to achieve the safety quality attribute the impact median value shows to what level or extent it supports or hinders the achievement of the system quality attributes according to the pre-defined scale for impact level.

Balancing the relationships between a software system's quality attributes is done in two steps. Firstly, by evaluating the impact level of the utilized architectural tactics to achieve a given quality attribute on other quality attributes achievement. Secondly, by selecting the appropriate architectural tactic in which it helps in sustaining a high quality level for a software system.

It is important to understand that the adoption of a certain architectural tactic used to achieve a certain quality attribute of a system can affect the relationship between the system quality attributes. Moreover, the safety quality attribute supports the achievement of the portability quality attribute of a software system, because safety candidate architectural tactics aim at maintaining the quality sub-characteristics of the portability quality attribute for the system by making the system adhere to application related standards. Additionally, the safety candidate architectural tactics can also be adjusted according to the systems' portability quality attribute conventions or regulations and similar prescriptions.

By comparing the results of the aforementioned recent researches with this research results, it can be seen that the relationships of quality attributes can be assessed differently. For example, as illustrated in Tables 6 and 7 the strength of the relationship created from focusing on achieving safety to the system produced a minor (i.e. +1) supporting relationship that may support the achievement of the maintainability quality attribute. On the other hand, when focusing on achieving maintainability the relationship, it is more supportive to sustain a higher level of safety to a software system. Therefore, software architects may sustain a sufficient level of quality to other software system while they are working on their way in achieving other quality attributes. However, in this research, the relationship between quality attributes is based

on the used solutions that are adopted to achieve a certain quality attribute. In the other research studies, the results are more generalized and are based on experiences from different views.

6- DISCUSSION AND VALIDATION

Usually, the achievement of some important quality attributes such as dependability is left until the late stages of the software development. This research focuses on manifesting the desired quality attributes requirements and concerns at earlier phases such as at the software architecture phase. In this phase, we have studied the methodologies that can be used as techniques to engineer in quality attributes to a software system. This method is based on two pieces of information architectural strategy (tactics) as candidate solutions and scenarios depicting all the necessary elements of a certain quality attribute. The understanding of quality attributes relationships is one of the factors that steer software development toward success. Therefore, we have developed a framework to understand the relationships between quality attributes based on the impact evaluation of the used tactics and relevant software quality requirements. We have taken two different quality attributes separately and evaluated their candidate solutions' impact on other quality attributes.

The quality attributes' reasoning framework is helpful to evaluate the quality attributes' impact in case a solution exists. Furthermore, this framework can only assess the impact when there are enough specifications of the developed system and its quality requirements. It also gives the opportunity to assess the architectural tactics concerning the system's desired quality attributes before using them. Moreover, the framework may produce a neutral relationship between the quality attribute in focus and the compared quality attributes when the number of the candidate architectural tactics of the quality attribute in focus which have a positive impact are equal to those which have a negative impact on the compared quality attributes.

The degree which describes the level of impact is dependent on the extent to what a given quality attribute candidate architectural tactic affects the achievement of other quality attributes' sub-characteristics (criteria), as most of the participants of the research's second survey wrote in their motivations explaining the base of their evaluation. Thus, the tactics impact evaluation can be undertaken at two levels:

1. As quality attributes consist of several sub-characteristics [7]. The first level describes the extent of hindering or supporting the compared quality attributes' sub-characteristics when using a given tactic.
2. The second level of impact evaluation is the level by which this tactic is hindering / supporting the achievement of the compared quality attributes.

The validation of our results is based on the possible validity threats discussed by Wohlin et al. [29]. In this research, the possible internal validity threats are carefully treated. Moreover, a stratified method as the sampling method has been used as it reduces the amount of errors in the process of choosing study subjects / participants [29]. Furthermore, the authors classified the subjects / participants into different single groups according to their profiles considering the possible validity threats to single groups, such as 'statistical regression and selection'[29].

In this research, survey was used as an approach to collect qualitative data to enable the researchers to produce quantitative data, and thus generate indicators to identify the nature and strength of the relationships between a two quality attributes with other quality attributes as stated in the ISO 9126 quality model [16] based on the impact evaluation of the proposed dependability quality attribute tactics. The ISO 9126 quality model is selected because it is widely accepted and used in industry [1, 27]. Furthermore, this quality model encloses the majority of quality attributes in different models such as McCall and Boehm [1, 7, 27].

7- CONCLUSION

When developing a software system, it is important to understand the impact of the used strategy or tactic on other desired quality attributes at earlier phases of the development process. Otherwise, more effort may be spent on trying to satisfy the system requirements. Conversely, the system development maybe facilitated by satisfying a quality attribute using a certain architectural tactic, and other quality attributes which are likely to follow because of the positive impact the used architectural tactic has.

The execution of this framework provides indicators to define the relationship between systems quality attributes by quantifying the impact of available solutions. The two main contributions of this research are focused on increasing the likelihood of sustaining a high level of quality to software system by providing thorough analysis for the quality attributes' relationship based on the available solution that could be illustrated from literature and experience. This research results also provide yet another factor for software architects to take into account when deciding on certain quality attributes candidate architectural tactics prioritisation; what, and in what order.

The development of the framework is focused on understanding the relationships between quality attributes based on the used solutions and systems' quality requirements. The quality attribute relationships reasoning framework helps in understanding the importance of making the right design decision embracing system quality requirements and other constraints such as operational and environmental. The evaluated impact of these solutions will help in making the proper selection of tactics to sustain a sufficient level of quality to

a software system and therefore satisfy one of the most important challenges in engineering quality to a software system. Moreover, we conclude that, when utilizing a tactic for developing a system, software architects must take into account to evaluate their architectural tactics or strategies according to different aspects, such as impact on achieving system's quality attributes, cost, implementation applicability to system context, and other factors as well as investigating the suitability of selected architecture pattern and style.

However, in order to have an accurate and a thorough assessment on the relationship between a software system quality attributes, software architects are required to develop more concrete scenarios to embrace all the non functional requirements in which their influence have a significant impact on the quality attributes in focus.

REFERENCES

- [1] R.E. Al-Qutash, "Quality Models in Software Engineering Literature: An Analytical and Comparative Study," *Journal of American Science*, 6:(3), 2010, pp. 166-175.
- [2] C. Andersson, *Managing Software Quality Through Empirical Analysis of Fault Detection*, Ph.D. Dissertation No. 1101-3931, Department of Communication Systems, Lund University, Skoene, Lund, Sweden, 2006.
- [3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, 1:(1), 2004, pp. 11-33.
- [4] F. Bachmann, L. Bass, and R. Nord, *Modifiability Tactics*, Tech. Report, TR-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2007.
- [5] L. Bass, M. Klein, and G. Moreno, *Applicability of General Scenarios to the Architecture Tradeoff Analysis Method*, Tech. Rep. TR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2001.
- [6] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd Ed., Addison Wesley Professional, New York, USA, 2003.
- [7] P. Berander, L. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jansson, S. Kogström, D. Milicic, F. Mrtensson, K. Rnkk, P. Tomaszewski, L. Lundberg, M/ Mattsson, and C. Wohlin, *Software Quality and Trade-Offs*, Tech. Report, Blekinge Institute of Technology, Ronneby, Blekinge, Sweden, 2005.

- [8] J. Bosch and P. Molin, P. "Software Architecture Design: Evaluation and Transformation," Proceedings of the IEEE Conference and Workshop on Engineering of Computer-Based Systems (ECBS'99), Nashville, TN, USA: March 7-12, 1999, pp. 4-10.
- [9] J. Bosch and L. Lundberg, "Software Architecture Engineering quality attributes," *Journal of Systems and Software*, 66:(3), 2003, pp. 183-186.
- [10] M. S. Deutsch and R.P. Willis, *Software Quality Engineering a Total Technical and Management Approach*, Prentice Hall, New Jersey, USA, 1988.
- [11] C. Ebert and R. Dumke, "Assuring the Quality of Software Systems" in *Software Measurement: Establish, Extract, Evaluate, Execute*, (Ebert C. and Gumke, R., Editors), Springer, Germany, 2007, pp. 230-300.
- [12] J. Heit, *Impact of Methods and Mechanisms for Improving Software Dependability on Non-Functional Requirements*, Ph.D. Dissertation No. 2548, Department of Informatics, Stuttgart University, Stuttgart, Germany, 2007.
- [13] K. Henningson and C. Wohlin, "Understanding the Relations between Software Quality Attributes: A Survey Approach," Proceedings of the 12th International Conference on Software Quality (ICSQ'02), Ottawa, ON, Canada: 2002, pp. 1-12.
- [14] IEEE, IEEE Std. 610.12: IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, New York, USA, 1990.
- [15] IEEE, ANSI/IEEE 1471: Recommended Practice for Architecture Description of Software-Intensive Systems, IEEE Computer Society Press, New York, USA, 2000.
- [16] ISO, ISO/IEC 9126-1: Software Engineering - Product Quality - Part 1- Quality Model, International Organization for Standardization, Geneva, Switzerland, 2001.
- [17] ISO, ISO/IEC 42010: Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems, International Organization for Standardization, Geneva, Switzerland, 2007.
- [18] R. Kazman, S.J. Carrière, and S.G. Woods, "Toward a Discipline of Scenario-based Architectural Engineering," *Annual of Software Engineering*, 9:(1-4), 2000, pp. 5-33.
- [19] N. Lassing, D. Rijsenbrij, and H.V. Vliet, "On Software Architecture Analysis of Flexibility, Complexity of Changes: Size isn't enough," Proceedings

- of 2nd Nordic Software Architecture Workshop (NOSA'99), Ronneby, Sweden: 1999, pp. 1103-1581.
- [20] J.A. McCall, Quality Factors, in *Encyclopaedia of Software Engineering*, (Marciniak, J., Editor), John Wiley & Sons, New York, USA, 1994, pp. 958-969.
- [21] K.B. Misra, Dependability Considerations in the Design of a System, in *Handbook of Performability Engineering*, (Misra, K. B., Editors), Springer, Germany, 2008, pp. 71-80.
- [22] S.R. Rakitin, Software Verification and Validation for Practitioners and Managers, 2nd Ed., Artech House Inc., Norwood, MA, USA, 2001, pp. 170-230.
- [23] T. Sherman, Quality Attributes for Embedded Systems, in *Advances in Computer and Information Sciences and Engineering*, (Sobh, T., Editor), Springer, Germany, 2008, pp. 535-539.
- [24] SSC, Smart 1: Technical Information, Swedish-Space-Cooperation, [online] <http://www.ssc.se/?id=6217>, retrieved on April, 2009.
- [25] M. Svahnberg, Supporting Software Evolution: Architecture Selection and Variability, Ph.D. Dissertation No.2003:03, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Ronneby, Blekinge, Sweden, 2003.
- [26] M. Svahnberg, C. Wohlin, L. Lundberrg, and M. Mattsson, A Quality Driven Decision Support Method for Identifying Software Architecture Candidates, *International Journal of Software Engineering and Knowledge Engineering*, 13:(5), 2003, pp. 547-573.
- [27] M. Svahnberg and K. Henningsson, "Consolidating Different Views of Quality Attributes Relationships," *Proceedings of the 7th ICSE Workshop on Software Quality (WOSQ'09)*, Vancouver, BC, Canada: May 16, 2009, pp. 46-50.
- [28] H.V. Vliet, *Software Engineering Principles and Practice*, 2nd Ed., John Wiley & Sons, New York, USA, 2000.
- [29] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*, 1st Ed., Kluwer Academic Publisher, Massachusetts, USA, 2000.
- [30] W. Wu and T. Kelly, "Safety Tactics for Software Architecture Design," *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, Hong Kong, China: 2004, pp. 368-375.

- [31] L. Zhu, M.A. Babar, and R. Jeffery, "Mining Patterns to Support Software Architecture Evaluation," Proceedings the 4th IEEE/IFIP Working International Conference on Software Architecture (WICSA'04), Oslo, Norway: June 12-15, 2004, pp. 25-34.
- [32] H. Zulzalil, A.A. Abdul Ghani, M.H. Selamat, and R. Mahmod, "A Case Study to Identify Quality Attributes Relationships for Web-based Applications," International Journal of computer Science and Network Security, 8:(11), 2008, pp. 215-220.

