A Tool for Trade-off Resolution on Architecture-Centered Software Development

Italo Carlo Lopes Silva  
Federal University of Alagoas  
italocarlo@nti.ufal.br

Patrick H S Brito  
Federal University of Alagoas  
patrick@ic.ufal.br

Baldoino Fonseca dos Santos Neto  
Federal University of Alagoas  
baldoino@ic.ufal.br

Evandro Costa  
Federal University of Alagoas  
evandro@ic.ufal.br

Hemilis Rocha  
Federal University of Alagoas  
joyseisel@gmail.com

ABSTRACT

The success of a software project is strongly related with architectural design. However, designing the right Software Architecture is a very subjective task and takes a long time, being much influenced by architect’s experience and the quality of requirements engineering. A big problem emerges when the trade-off among quality requirements is not properly solved during requirements engineering. The objective of this paper is to present a trade-off resolution process supported by a tool for helping young requirements engineers and software architects on the hard task of specifying the system requirements, detecting and solving trade-offs among them.

Keywords

Requirements engineering, Trade-off resolution, Software architecture

1. INTRODUCTION

Develop software with low cost of production and maintenance, to be delivered in a shorter time with requirement of high quality has become even more urgent and necessary. In this way, one of the key elements of success for the software projects is the architectural design.

Software architectures represent explicitly the structure of systems, and it is one of the earliest artifacts that permits the analysis of system quality attributes, such as dependability, including reliability, availability, security and safety [1].

Although each system has its particularity, it is common that applications that belong to the same domain are to use similar architecture. In this way, the architectural styles are very important, because they define a family of systems in terms of a pattern of structural organization, thus providing meta-models that can be applied in recurrent problems of a domain [5]. However, designing the right Software Architecture is a very subjective task and takes a long time, being much influenced by architect’s experience as well as the quality of requirements engineering. A big problem emerges during the architectural design phase when trade-offs amongst quality attributes have not been identified and properly managed during requirements engineering.

Each alternative solution satisfies different functional and non-functional requirements to varying extents. Selecting a solution among multiple alternatives involves managing trade-offs among requirements, with respect to stakeholders preferences and consequences of alternatives on the requirements [2].

The main problems when managing trade-offs among requirements and deciding over alternative design solutions are [2]: (1) Extensive Data Collection, (2) Manual prioritization, (3) Incomparable Scales and (4) Scalability.

The objective of this paper is to present a trade-off resolution process with tool support for helping young requirements engineers and software architects on the hard task of specifying the system quality requirements, detecting and solving trade-offs among them. The tool has a rule-based architecture which uses an expert system engine for identifying trade-offs and provide directions for managing them. The rules database can be seen as a knowledge repository that contains technical solutions based on patterns and good experiences of specialists.

The proposed tool is presented in details, including its requirements, its software architecture and detailed design, as well as the structure of the rules for representing the specialist’s knowledge.

The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 presents the trade-off management process proposed in this paper. Section 4 presents the tool developed for supporting the aforementioned process. Finally, Section 5 presents some concluding remarks and directions of future work.

2. RELATED WORK

Faced with the typical absence of reliable quantitative data, some Requirements Engineering (RE) techniques, such as Tropos [6] and i* [15] treat quality attributes as soft goals, thus enabling to reason about the partial satisfaction of such goals by using qualitative labels such as partially satisfied, sufficiently satisfied, partially denied, and fully denied. Although it is considered a good evolution, the subjectivity of classification and evaluation of quality attributes could produce conflicting quality requirements. When conflicts emerge among quality attributes, the management of the trade-offs becomes a critical issue [13].

Elahi and Yu [2] present a semi-automated tool that uses Even Swaps process [7] for decision making related to conflicting requirements. Although this tool addresses the same focus of our approach, we focus on managing trade-off involving specifically software quality attributes. Besides, we
also aim to clarify the meaning of quality attributes and the relationship among them, in order to improve the confidence of the stakeholder on the final quality attribute specification.

García-Mireles et al. [4] present a conceptual framework for dealing with software quality trade-offs. Such solution presents a set of activities for managing trade-offs based on a systematic comparison of CMMI and ISO-12207 specifications. Our work is compliant with García-Mireles et al.’s approach and can be seen as a tool-supported instance of such conceptual framework.

3. A PROCESS FOR MANAGING TRADE-OFFS AMONG QUALITY ATTRIBUTES

The Trade-offs management is an important task, which can influence many activities of software development. In terms of quality attributes, the trade-off management is specially important in the context of architectural design. Figure 2 presents an overview of the proposed process, which also contextualizes the trade-off management activities with architectural design activities.

The first activity, named 01-Prepare the Environment focuses on the persistence of specialist’s knowledge related to requirements engineering. This activity should be executed at the beginning, after the deployment of the proposed solution, but can also be re-executed anytime, in order to update the knowledge database with expertises. The second activity, named 02-Specify Quality Requirements consists on the utilization of the proposed tool using the existing specialized knowledge. These activities will be detailed in the next subsections.

3.1 Registering the Quality Attributes

The first thing to be done in whole process is the register of quality attributes (Activity 01.1). In the context of this paper, we have registered the quality attributes proposed by ISO/IEC 9126 [3]. The main idea consists in using these information to collect user’s preferences about the desired quality requirements.

These registered information about the quality attributes will be used to collect the attributes’ priorities, thus the end users won’t need to know technical terms or details about a particular domain or technology. In this sense, we considered adopting a mapping between natural language and quality requirements, where the engineers and architects will associate each quality attribute with a text about this attribute. The end user must understand this text easily, reducing the effort to collect the priority of a particular requirement correctly.

Additional information such as features, advantages and disadvantages could be registered to improve the understanding about a specific requirement. Each registered quality attribute will be associated to an element in a knowledge repository. This association is very important, because once the priorities of requirements were filled (Activity 02.1), the tool will retrieve the elements which are associated with each attribute and will fill properly the working memory with their priorities.

3.2 Registering the Trade-off Patterns

After registering the quality attributes, it is necessary to identify potential trade-offs among them (Activity 01.2). The potential trade-offs, called patterns, consist on well known scenarios in which two or more quality attributes conflict each other. For example, on a web application, the software architect noticed that when the number of concurrent requests increases, the application becomes slower. This scenario could generate a pattern exemplifying a case when scalability and performance conflict each other.

The identification of such trade-off patterns can be done both based on previous experiences and on the specialized literature. In the context of this paper, we have registered trade-off patterns based on real scenarios reported by literature [11].

Once the stakeholder confirms the priorities for each quality attribute, they will be used as input to the working memory of trade-off management process (Activity 02.3), using for this the association made in the Register Quality Attributes activity (Activity 01.1), that will be responsible to identify and solve possible trade-offs among two or more requirements.

3.3 Registering Rules for Prioritize Requirements

Then, in order to finish the environment preparation, it is necessary to define rules to support trade-offs management (Activity 01.3). Such rules are described through a list of easy questions, whose answers reflect priorities over the requirements.

For example, in the context of the trade-off scenario of the hypothetical web application, a helpful question could be: "Would you limit the number of concurrent request in order to provide a better quality of service in terms of performance?". Each registered trade-off scenario should be associated to at least one question and each question should have an impact associated to its answers, which can increase or decrease the weight of the quality attributes, thus impacting on its priority, as can be seen in Figure 1 that shows the meta-model for representing the rules. For simplifying the diagram, most attributes have been omitted. It is important to emphasize that the association between TradeOffPattern and Domain allows the definition of either domain-specific and general-domain trade-off patterns.

3.4 Eliciting the Quality Requirements

Firstly, the activity named Specify Quality Requirements (Activity 02) starts with the stakeholder trying to identify the quality attributes which his system must meet (Activity 02.4). The end user will have to answer several easy, interesting questions about the desired quality. Each answer will be associated to a quality attribute, which we call the reference attribute, and to a weight for each answer. The weight is used to determine which attributes are more important than others in the system.

Once the stakeholder confirms the priorities, he will be able to register his preferences, and these preferences will be used to select the appropriate elements from the knowledge base. Each registered trade-off will be associated to one or more quality attributes, and each registered trade-off will be associated to one or more elements from the knowledge base.

After registering all trade-offs, the stakeholder will be able to register his preferences, which will be used to select the appropriate elements from the knowledge base. Each registered trade-off will be associated to one or more quality attributes, and each registered trade-off will be associated to one or more elements from the knowledge base.
The tool will retrieve all the quality requirements persisted in the database and will show a list with information about each one of attributes that were registered previously (Activity 01.1).

This tool will give 100 points to the stakeholder to distribute among the attributes which he is think that is important for his software. All attributes start with the same priority, that is zero. The importance of a requirement will be measured using the points associated for each attribute.

Once the stakeholder confirms the priorities for each quality attribute, they will be used as input to the working memory of trade-off management process (Activity 02.3), using for this the association made in the activity named Register the Quality Attributes (Activity 01.1), that will be responsible to identify and solve possible trade-offs among two or more requirements.

### 3.5 Understanding the Requirements

For each quality attribute selected, it is important to provide extra details in order to validate properly the chosen quality attributes (Activity 02.2). This activity will give support the previous activity (Activity 02.1), providing extra information about each one of registered attributes. Anytime, when the stakeholder has some doubt or the information is not sufficiently clear about a requirement, he will can use this resource to clarify any misunderstood about this item. Such details can be, for example, a detailed description of the quality attribute with an illustrating example, its features, advantages and disadvantages.

### 3.6 Managing Trade-offs

Finally, after having the real list of quality attributes aimed for the application, might be necessary to manage trade-offs among them (Activity 02.3). For this, it is necessary to follow five sequential steps: (1) Identify trade-off patterns among the selected quality attributes; (2) Ask questions related to the identified trade-off patterns; (3) Compute answer’s impact and update the weight attribute (positive and negative); (4) Sort quality attributes based on the weight attribute; (5) Classify requirements following a predefined taxonomy.

The taxonomy for classifying quality attributes aims to reduce subjective interference. Analyzing existing guidelines [9], we have designed a taxonomy for classifying quality attributes in four possible levels: (1) Mandatory, for those attributes whose bad functioning could preclude the use of the system; (2) Important, for those attributes that the stakeholder would be willing to pay more for them; (3) Desirable, for those attributes that the stakeholder wants, but would not be willing to pay more for them; and (4) Not Important, for those attributes that the stakeholder does not require, but can be implemented if do not impact on cost. After having the quality attributes sorted by importance (weight), the stakeholder should classify in the four categories proposed. A possible way for automating this task is to establish a threshold value of weight. But in case of keeping trade-off even after the aforementioned resolution process, the software architect should be warned about the problem and solve it manually.

### 4. A TOOL FOR SUPPORTING THE PROPOSED PROCESS

A generic trade-off resolution tool for supporting the process presented in Section 3 has been developed. We have used the Java programming language and MySQL database using JPA framework for persistence. During the tool specification, we opted to perform the two first activities of the execution process at the same time.

After selecting the desired quality attributes, the user should confirm in order to manage trade-offs. The initial weight of the quality attributes is zero, thus indicating that all of them initially have the same priority. Alternatively, the user can choose to manually set the initial priority. For this, 100 points are given to the user, who can distribute these points over the quality attributes. When the priorities were filled, all the information will be passed to a working memory of the expert system that manages trade-offs (Activity 02.3). After the trade-off management process, such importance order is updated, based on the final weight.

For providing more details about the design of tool, Figure 3 presents its software architecture, which follows an heterogeneous architectural style which combines data-centered and rule-based characteristics.
The ExpertSystem component consists on a rule-based system that emulates the decision-making ability of a human expert [10]. Such systems are designed to solve complex problems by reasoning about knowledge by using an inference engine, which is a computer program that tries to derive answers from a knowledge base. The developed tool reused an existing rule-based inference engine called Inabit [14, 12], an Intelligent Agent Building Tools. Inabit may be seen as a kind of rule-based expert system shell, but it may also be seen as a software framework by providing a set of facilities to be used by the developers of intelligent applications. Therefore, the main idea is to provide effective support with facilities to both domain expert/knowledge engineer and developer. Inabit provides various features commonly used in a typical expert system, such as (i) inference by using goal driven reasoning (backward chaining) or data driven reasoning (forward chaining), (ii) basic explanation facilities, (iii) facilities to create and edit a given knowledge base.

For this, The FormFiller component will have the task of passing the stakeholder’s answers regarding quality attributes to the ExpertSystem component to be inserted in the working memory of framework. With all these information, the Trade-offSolver component will use the framework, running the forward chaining reasoning in the knowledge stored to identify the trade-offs problems. Once these trade-offs were identified, the tool will try to solve them presenting questions associated to the trade-off patterns, as presented in Section 3.

5. CONCLUSION AND FUTURE WORK

This paper presented an approach to support the requirements engineering phase of a software. The proposed solution comprises a systematic process supported by a tool for helping young requirements engineers and software architects on the hard task of specifying the system requirements, detecting and solving trade-offs among them. The tool has a rule-based architecture which uses an expert system engine that behaves as a requirement engineer, thus keeping the technical knowledge and experience at the software company. Although other existing approaches address the same target, we focus on managing trade-off involving specifically software quality attributes. Besides, we also aim to clarify the meaning of quality attributes and the relationship among them, in order to improve the confidence of the stakeholder on the final quality attribute specification.

As an immediate future work, the approach should be evaluated in real scenarios and domains, with a high number of volunteers. Furthermore, it is also being evolved in order to link trade-off resolution of quality attributes to design decisions related to the choice of architectural styles during the architectural design phase. In addition to supporting the architectural design by recommending reference architectures, this integration also aims to increase the quality of trade-off resolution, since the trade-offs are better resolved when contextualized with design artifacts, such as software architecture and source-code [8].

6. REFERENCES