A requirements model for quality attributes

Isabel Brito[1], Ana Moreira[2], João Araújo[2]

[1]Instituto Politécnico de Beja, Beja, Portugal
isabel.sofia@estig.ipbeja.pt
[2]Departamento de Informática, FCT/UNL, Caparica, Portugal
{amm,ja}@di.fct.unl.pt

Abstract

Quality attributes can be assumptions, constraints or goals of stakeholders. In this paper we present a process to identify and specify quality attributes and to integrate them with functional requirements. The crosscutting nature of some of the quality attributes influences negatively, for example, reusability and traceability in the later stages of the software engineering process. To minimize that influence we start by proposing a template to specify quality attributes at the requirements stage. Then, we extend use cases and sequence diagrams to specify the integration of those attributes with functional requirements.

1 Introduction

Quality attributes, such as response time, accuracy, security, reliability, are properties that affect the system as a whole. Most approaches deal with quality attributes separately from the functional requirements of a system. This means that the integration is difficult to achieve and usually is accomplished only at the later stages of the software development process. Furthermore, current approaches fail in dealing with the crosscutting nature of some of those attributes, i.e. it is difficult to represent clearly how these attributes can affect several requirements simultaneously. Since this integration is not supported from requirements to the implementation, some of the software engineering principles, such as abstraction, localization, modularisation, uniformity and reusability, can be compromised.

What we propose is a model to identify and specify quality attributes that crosscut requirements including their systematic integration into the functional description at an early stage of the software development process, i.e. at requirements.

The rest of this paper is organised as follows. Section 2 presents a model for early quality attributes and discusses its main activities. Section 3 applies our approach to a case study. Finally, concluding remarks are given in Section 4.

2 A model for early quality attributes

The process model we propose is UML compliant and is composed of three main activities: identification, specification and integration of requirements. The first activity consists of identifying all the requirements of a system and select from those the quality attributes relevant to the application domain and stakeholders. The second activity is divided into two main parts: (1) specifying functional requirements using a use case based approach; (2) describe quality attributes using special templates and identify those that cut across (i.e. crosscutting) functional requirements. The third activity proposes a set of models to represent the integration of crosscutting quality attributes and functional requirements. Figure 1 depicts this model.

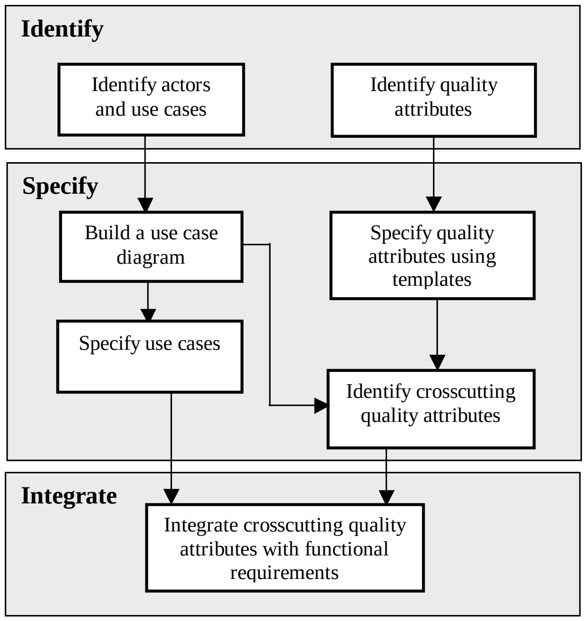
Identify
Identify actors and use cases
Identify quality attributes
Specify
Build a use case diagram
Specify quality attributes using templates
Specify use cases
Identify crosscutting quality attributes
Integrate
Integrate crosscutting quality attributes with functional requirements

Figure 1: A requirements model for quality attributes

The template we propose to specify a quality attribute was influenced by Mylopoulos *et al.* [8] and Bredmeyer [6, 7] (see Figure 2).

Name	The name of the quality attribute.
Description	Brief description.
Focus	A quality attribute can affect the system (i.e. the end product) or the development process.
Source	Source of information (e.g. stakeholders, documents).
Decomposition	Quality attributes can be decomposed into simpler ones. When all (sub) quality attributes are needed to achieve the quality attribute, we have an AND relationship. If not all the sub quality attributes are necessary to achieve the quality attribute, we have an OR relationship.
Priority	Expresses the importance of the quality attribute for the stakeholders. A priority can be MAX, HIGH, LOW and MIN.
Obligation	Can be optional or mandatory.
Influence	Activities of the software process affected by the quality attribute.
Where	List of models (e.g. sequence diagrams) requiring the quality attribute.
Requirements	Requirements describing the quality attribute.
Contribution	Represents how a quality attribute can be affect by the others quality attributes. This contribution can be positive (+) or negative (-).

Figure 2: Template for quality attributes

To identify the crosscutting nature of some of the quality attributes we need to take into account the information contained in rows Where and Requirements. If a quality attribute cuts across (i.e. is required by) several requirements and models, then it is crosscutting.

The integration is accomplished by "weaving" the quality attributes with the functional requirements in three different ways [1, 3, 5, 12]:

(1) Overlap: the quality attribute adds new behaviour to the functional requirements it transverses. In this case, the quality attribute may be required *before* those requirements, or, it may be required *after* them.

(2) Override: the quality attribute superposes the functional requirements it transverses. In this case, its behaviour substitutes the functional requirements behaviour.

(3) Wrap: the quality attribute "encapsulates" the requirements it transverses. In this case the behaviour of the requirements is wrapped by the behaviour of the quality attribute.

We weave quality attributes with functional requirements by using both standard diagrammatic representations (e.g. use case diagram, interaction diagrams) and by new diagrams.

3 Applying the approach to a case study

The case study we have chosen is a simplified version of the toll collection system implemented in the Portuguese highways [2].

"In a road traffic pricing system, drivers of authorised vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lanes. A driver has to install a device (a gizmo) in his/her vehicle. The registration of authorised vehicles includes the owner's personal data, bank account number and vehicle details.

A gizmo is read by the toll gate sensors. The information read is stored by the system and used to debit the respective account. When an authorised vehicle passes through a green lane, a green light is turned on, and the amount being debited is displayed. If an unauthorised vehicle passes through it, a yellow light is turned on and a camera takes a photo of the plate.

There are green lanes where the same type of vehicles pay a fixed amount (e.g. at a toll bridge), and ones where the amount depends on the type of the vehicle and the distance travelled (e.g. on a motorway)."

3.1 Identify requirements

Requirements of a system can be classified into functional and non-functional (i.e. quality attributes). Functional requirements are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. Different types of methods are used to specify functional requirements. Use case driven approaches describe "the ways in which a user uses a system" that is why use case diagram is often used for capturing functional requirements [9]. Quality attributes define global properties of a system. Usually these are only dealt with in the later stages of a software development process, such as design and implementation.

Identify actors and use cases.
For the road pricing system, the actors we identified are:

- Vehicle owner: is responsible for registering a vehicle;

- Vehicle driver: comprehends the vehicle, the driver and the gizmo installed on it;
- Bank: represents the entity that holds the vehicle owner's account;
- System clock: represents the internal clock of the system that monthly triggers the calculation of debits.

The following are the use cases required by the actors listed above:

- Register vehicle: is responsible for registering a vehicle and its owner, and communicate with the bank to guarantee a good account;
- Pass single toll: is responsible for dealing with tolls where vehicles pay a fixed amount. It reads the vehicle gizmo and checks on whether it is a good one. If the gizmo is ok the light is turned green, and the amount to be paid is calculated and displayed. If the gizmo is not ok, the light is turned yellow and a photo is taken.
- Enter motorway: checks the gizmo, turns on the light and registers an entrance. If the gizmo is invalid a photo is taken and registered in the system.
- Exit motorway: checks the gizmo and if the vehicle has an entrance, turns on the light accordingly, calculates the amount to be paid (as a function of the distance travelled), displays it and records this passage. If the gizmo is not ok, or if the vehicle did not enter in a green lane, the light is turned yellow and a photo is taken.
- Pay bill: sums up all passages for each vehicle, issues a debit to be sent to the bank and a copy to the vehicle owner.

Identify quality attributes.
Quality attributes can be assumptions, constraints or goals of stakeholders. By analysing the initial of set requirements, the potential quality attributes are identified. For example, if the owner of a vehicle has to indicate, during registration, his/her bank details so that automatic transfers can be performed automatically, then security is an issue that the system needs to address. Another fundamental quality attribute is response time that is a issue when a vehicle passes a toll gate, or when a customer activates his/her own gizmo in an ATM: the toll gate components have to react in time so that the driver can see the light and the amount being displayed. Other concerns are identified in a similar fashion: Multi-user System, Compatibility, Legal Issues, Correctness and Availability.

3.2 Specify functional requirements and quality attributes

The functional requirements are specified using the UML models, such as use cases, sequence and class diagrams. The quality attributes are described in templates of the form presented in Figure 2.

Build the use case diagram.
The set of all use cases can be represented in a use case diagram, where we can see the existing relationships between use cases and the ones between use cases and actors. Figure 3 shows the use case diagram of the road traffic system.

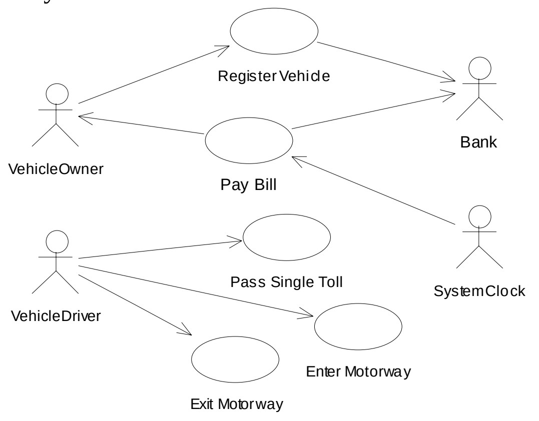

Figure 3. The use case diagram of the Road Traffic Pricing System

Later versions of the use case diagram can show relationships between use cases, in particular some of the use cases share a common set of events in the beginning (which could be shown by adding an extra use case related to the original use cases with the "include" relationship). Extend relationship could also be applied to deal with error situations, for example.

Specify use cases.
Use cases can be described using several techniques, ranging from natural language, scenarios, to formal representations. We prefer to use scenarios, described as a list of numbered steps [10]. A scenario is a particular path of execution through a use case. Use cases can be fully described using a primary scenario and several secondary scenarios, depending on the use case complexity. The primary scenario represents the main path of the use case, i.e. the optimistic view. The secondary scenarios describe alternative paths, including error conditions and exception handling. Each scenario can then be better described using a sequence diagram. A sequence diagram shows the temporal order of interactions between the objects involved in a scenario.

In our case study, we can identify at least two scenarios for each use case. For example, each of the use cases *Pass Single Toll*, *Enter Motorway* and *Exit Motorway*, has a scenario to deal with authorised vehicles and another to deal with non-authorised vehicles. Figure 4 shows the primary scenario "pass single toll gate ok".

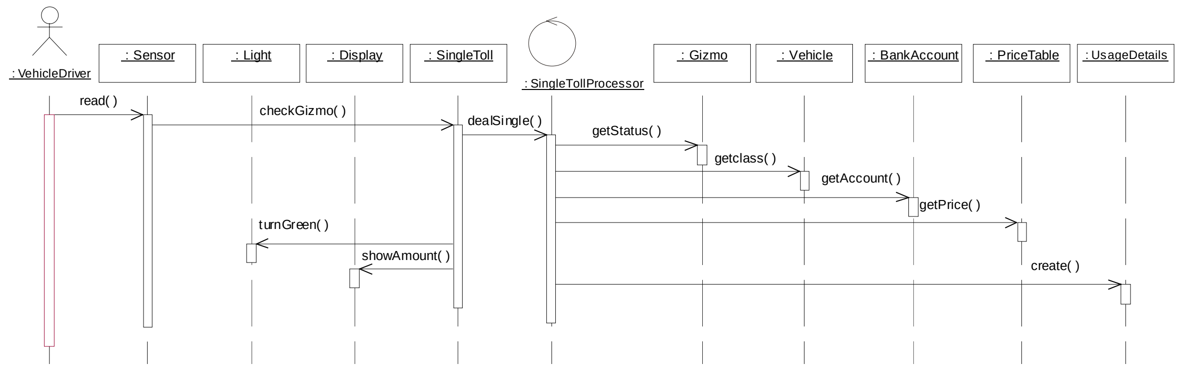

Figure 4. Sequence diagram for primary scenario of *PassSingleToll*

By building a sequence diagram, we can identify the objects in the system needed to handle the corresponding scenario. In our case, we have identified the interface object SingleToll, composed of Display, Light and Sensor, the entity objects Gizmo, Vehicle, BankAccount, PriceTable and UsageDetails and the control object SingleTollProcessor. The actors use the interface objects to interact with the system. The entity objects represent the core entities of the system, i.e. the objects with data. Finally, the control objects are the decision makers, i.e. the objects that decide what should be done next.

Specify quality attributes.
To specify quality attributes we use the template presented in Section 2. Let us describe the response-time and security quality attributes (see Figures 5 and 6).

Name	Response Time
Description	Period of time in which the system has to respond to a service
Focus	System
Source	Stakeholders, original description of the problem
Decomposition	<none>
Priority	MAX
Obligation	Mandatory
Influence	Design, system architecture and implementation
Where	Actors: VehicleDriver, Bank Use cases: RegisterVehicle, PassSingleToll, PassExitToll and PassEntryToll
Requirements	1. A toll gate has to react *in-time* in order to: 1.1 read the gizmo identifier; 1.2 turn on the light (to green or yellow) before the driver leaves the toll gate area; 1.3 display the amount to be paid before the driver leaves the toll gate area; 1.4 photograph the unauthorised vehicle's plate number from the rear. 2 The bank has to react *in-time* when customers (re) actives their gizmos through an ATM.
Contribution	(-) to security and to multi-user

Figure 5. Template for Response Time

As can be detected from the templates depicted in Figures 5 and 6, there is a priority conflict between response time and security. A trade-off must be negotiated with the stakeholders to resolve this conflict.

Identify crosscutting quality attributes.
If a quality attribute affects more than one use case (see *Where*), it is crosscutting. This can also be confirmed by analysing the *Requirements* row. For example "response time" cuts across PassSingleToll, PassExitToll and PassEntryToll.

3.3 Integrate functional requirements with crosscutting quality attributes

Integration composes the quality attributes with the functional requirements, to obtain the whole system. We use UML diagrams to show the integration. The two examples given above (for response time and security) fall into two of the categories already described: overlap and wrapper. We could extend the UML diagrams to represent some quality attributes. For example, the sequence diagram shown in Figure 4 can be extended to show how response time affects a scenario (see Figure 7).

Name	Security
Description	Restricts the access to the system and to the data within the system
Focus	System
Source	Stakeholders
Decomposition	Integrity and Confidentiality. Both have a AND relationship with Security
Priority	MAX
Obligation	Mandatory
Influence	Design, system architecture and implementation
Where	Actors: VehicleOwner, Bank Use cases: PayBill, RegisterVehicle
Requirements	The system must: 1. protect the vehicle's owner registration data 2. guarantee integrity in the data transmitted to the bank 3. guarantee integrity on data changed/queried by the operator 4. …
Contribution	(-) to response time and (+) to multi-user and compatibility.

Figure 6. Template for Security

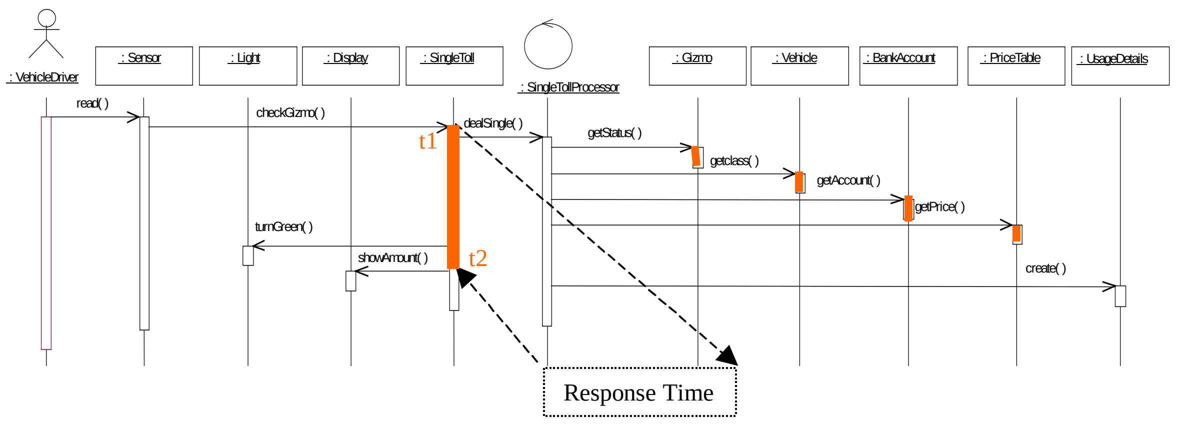

Figure 7. Response Time quality attribute wrap the functional requirements

This figure represents response time wrapping some functional requirements in the sequence diagram. The arrows and the grey rectangles identify the points wherein the constraint applies.

Figure 8 represents security crosscutting use cases. Remember that security is decomposed into confidentiality and integrity. This is an example of an overlap situation, where confidentiality must be used before the execution of the use cases and integrity must be used after it. Dashed lines are used to represent the <<after>> condition and dark lines represent the <<before>> condition.

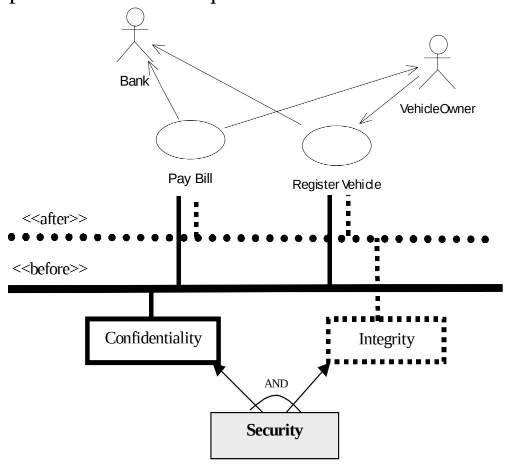

Figure 8. Security overlaps use cases

The (sub) qualities have an AND relationship with Security (see Figure 8).

4. Concluding remarks and future work

This paper presented an approach to identify, specify and integrate requirements. First we identify all the requirements of a system and select from those the quality attributes relevant to the application domain and stakeholders. Afterwards, we specify functional requirements, using a use case based approach, and describe quality attributes using special templates, identifying those that crosscut functional requirements. Finally, we propose a set of models to represent the integration of crosscutting quality attributes with functional requirements.

Last year's major conferences on software engineering and requirements engineering have published some interesting work on quality attributes and crosscutting concerns. Part of our future work is to investigate how other approaches such as ATAM (*Architecture Tradeoff Analysis Method*), composition patterns and goal-oriented requirements engineering relate to our work.

References

[1] Bergmans, L. M. J. and Aksit, M.. "Composing Software from Multiple Concerns: A Model and Composition Anomalies. Multi Dimensional Separation of Concerns in Software Engineering Workshop, ICSE 2000, Limerick, Ireland, 2000.

[2] Clark, R. and Moreira, A. "Constructing Formal Specifications from Informal Requirements", in proc. Software Technology and Engineering Practice, IEEE Computer Society, Los Alamitos, California, 1997, pp. 68-75.

[3] Clarke, S., Walker, R.J. "Composition Patterns: An Approach to Designing Reusable Aspects". In Proceedings of International Conference On Software Engineering (ICSE 2001), Toronto, Canada., 2001.

[4] Constantinides, C. A., Bader, A. and Elrad, T. *An Aspect-oriented Design Framework*. ACM Computing Surveys, March 2000.

[5]IBM Research, MDSOC Software Engineering using Hyperspace http://www.research.ibm.com/hyperspace/

[6] Malan, R., and Bredemeyer, D., "Defining Non-Functional Requirements", http://www.bredemeyer.com/papers.htm

[7] Malan, R. and Bredemeyer, D., "Functional Requirements and Use Cases", http://www.bredemeyer.com/papers.htm

[8] Mylopoulos, J., Chung, L., and Nixon, B., "Representing and Using Non-Functional Requirements: A Process-Oriented Approach", *IEEE Transactions on Software Engineering, Special Issue on Knowledge Representation and Reasoning in Software Development*, Vol. 18(6), 1992, pp. 482-497.

[9] Sommerville, I. and Sawyer, P., " Requirements Engineering, A good practice guide", John Wiley and Sons, 2000.

[10]Schneider G. and Winters J., *Applying Use Cases – A Practical Guide*, Addison-Wesley, 1998.

[11] Suzuki, J. and Yamamoto, Y.. "Extending UML with Aspects: Aspect Support in the Design Phase". AOP Workshop at ECOOP'99, Lisbon, Portugal, 1999.

[12] Xerox PARC, *AspectJ*. http://www.aspecj.org.