A Framework for Multi-Layered Requirements Documentation and Analysis

Michel dos Santos Soares
Computing Faculty
Federal University of Uberlândia
Uberlândia, Brazil
michel@facom.ufu.br

Jos Vrancken
Systems Engineering Section
Delft University of Technology
Delft, The Netherlands
j.l.m.vrancken@tudelft.nl

Abstract—The complexity of current software-intensive systems must be tackled during all phases of the system life cycle. Requirements Engineering is considered both by practitioners and researchers a crucial phase of software development. This article tackles requirements complexity by proposing a framework to document and analyze requirements at different levels of detail, inspired by the layered style of software architecture. Therefore, separation of concerns, an important concept used for software design, is applied in the context of requirements documentation and analysis. The framework has been applied in practice to design software-intensive systems in many domains. Specifically for this article, the proposal is to apply the framework to document and to analyze requirements for road traffic management systems.

Keywords—Requirements documentation; Requirements Analysis; SysML; Software-Intensive Systems

I. INTRODUCTION

Requirements Engineering is an area of research that involves a number of activities such as elicitation, validation, documentation, and management of requirements [21]. When any of these activities are poorly performed, the software project is at risk of failure [1]. According to a number of practitioners and researchers, Requirements Engineering is the most problematic phase of the development of a software system [18] [10] [20]. According to Brooks [8], knowing what to build, which includes requirements elicitation and specification, is the most difficult phase in the design of software.

Requirements are often misinterpreted, misunderstood, and poorly documented [4], [20]. A major issue is that requirements are often written at only one level of abstraction, or requirements at different levels of abstraction are mixed, which brings even more confusion to stakeholders. Requirements specifications for large systems should be specified at a number of levels of abstraction, as for instance, user requirements and systems requirements [28]. Another issue is that the majority of modeling languages are tailored to document requirements at only one level of abstraction. For instance, high level user requirements are often modeled using UML Use Cases. However, Use Cases are unsuitable to model more specific system level requirements [24], and can neither express structure between use cases nor a structural hierarchy of use cases in an easy and straightforward way [14].

Another common problem for software development is the poor relationship between the requirements engineering and the software architecture teams [15]. It is of utmost importance that both groups work together [5]. The software architecture must be considered when engineering new requirements to be implemented in a software product [19]. Requirements influence the architectural style to be chosen and the decisions that tailor the architecture. On the other hand, each architectural style is more or less capable of addressing a different number of non-functional requirements. Therefore, requirements engineering activities must be closely related to the design of the software architecture.

Separation of concerns and reasoning on multiple abstract levels are considered good principles of Software Engineering [7] and scientific knowledge in general [11]. This article proposes a framework (Section 3) for requirements in which these concepts are taken into account, after briefly evaluating a (non-exhaustive) number of techniques, methods, and languages used in Requirements Engineering activities (Section 2). The framework address the documentation and analysis of requirements at different levels of detail, inspired by the layered style of software architecture [2]. The framework has been applied in practice (Section 4) to a number of software projects in the field of road traffic control.

II. EVALUATION OF APPROACHES FOR REQUIREMENTS ENGINEERING

There are several approaches to modeling requirements. Basically, these approaches can be classified as graphics-based, purely textual, or a combination of both. Some are generic while others are part of a specific methodology. A non-exhaustive list of techniques used for requirements modeling and analysis is presented in the following paragraphs.

Natural language is the most common approach to write user requirements. The advantage is that natural language is the main mean of communication between stakeholders. However, problems such as misunderstandings, imprecision, ambiguity and inconsistency are common when natural language is used [17].

Structured Natural languages are used with the purpose of giving more structure to requirements documents [9].
Nevertheless, structured natural languages are neither formal nor graphical, and can be too much oriented to algorithms and specific programming languages. Other collateral effects are that structured specifications may limit too early the programmers’ freedom, and are mostly tailored towards procedural languages, being less suitable for modern languages and paradigms.

User Stories have been used as part of the eXtreme Programming (XP) [3] agile methodology. They can be written by the customer using non-technical terminology, in the format of sentences using natural language. Although XP offers some advantages in the Requirements Engineering process in general, such as user involvement and defined formats for user requirements and tasks, requirements are still loosely related, not graphically specified, and oriented to a specific methodology.

A well-known diagram used for requirements modeling is the Use Case diagram. Even before UML emerged as the main Software Engineering modeling language, Use Cases were already a common practice for graphically representing functional requirements in methodologies such as the Object-Oriented Software Engineering (OOSE) [16]. Use Cases have some disadvantages and problems [24]. They are applied mainly to model functional requirements and are not very helpful for other types of requirements, such as non-functional ones [25].

Two SysML diagrams are distinguish as useful mainly for Requirements Engineering activities: the SysML Requirements diagram and the SysML Use Case diagram [23]. One interesting feature of the SysML Requirements diagram is the possibility of modeling other type of requirements besides the functional ones, such as non-functional and external requirements. The SysML Use Case diagram is derived from the UML Use Case diagram without important modifications. In addition to these diagrams, SysML Tables can be used to represent requirements in a tabular format. Tabular representations are often used in SysML but are not considered part of the diagram taxonomy [23].

The framework proposed is based on a combination of UML/SysML Use Cases, SysML Tables, and SysML Requirements diagrams. The main reasons for this choice are:

- The use of graphical modeling languages to model requirements.
- The use of already well-known modeling languages commonly applied in academia and in industry.
- Both SysML and UML are described by the same metametamodel, the MOF [22]. Therefore, software developers that are familiar with UML will have a stepper learning curve.
- Both SysML and UML are extensible modeling languages.

III. PROPOSED FRAMEWORK

As discussed in [2], different stakeholders of an architecture will need different levels of detail, i.e., depending on their needs, they are interested in more abstract or more specific information. The framework presented in this section takes this concept of different levels of abstraction to be used in Requirements Engineering. The proposed framework can be seen as a guideline to create software requirements documents.

The document of requirements is an artefact normally produced during the early stages of software development and maintained through the development process. The document will be read by a number of stakeholders, each one with different concerns. These concerns are well-separated at different levels using the proposed framework described in this section.

A. Modeling at different levels of abstraction

![Figure 1. Requirements framework](image)

Important characteristics of the framework are separation of concerns, the possibility of identifying and modeling requirements at different levels of abstraction, hiding details that are not necessary for a specific stakeholder, and addressing complexity. Given these characteristics, the layered style [13] for software architecture was chosen to create the framework. The style presents a number of well-known advantages, such as design based on increasing levels of abstraction, support for reuse, and scalability. Three layers for requirements are proposed in this framework (Figure 1).

The top layer is about documenting user requirements. These requirements are derived directly from users and clients. User requirements are high-level, business oriented requirements. The important concern at this level is to document “what” the system will do, i.e., the functionalities and constraints from the user point of view. The most common approach to be used at this level is to document requirements using natural language.

The middle layer deals with system (software) requirements. The purpose of requirements at this level is to be a refinement of user requirements, with increasing levels of detail. These requirements are used as input for designing
software using diagrams of a modeling language such as UML [6].

The bottom layer deals with implementation requirements. At this level, requirements must be specified in a very detailed manner. The level of detail is close to algorithm specifications. Examples of tools for this level are UML Activity diagrams, decision tables, and pseudocode.

B. Relationship between Requirements

The relationship between requirements is also documented in this framework. With this proposed approach, once a user requirement is modified, it traces forward to system-level and implementation-level requirements. Thus, software designers know which requirements are affected by changing user requirements. Similarly, when changing implementation requirements or systems requirements, it is possible to trace back which user requirements are affected. Considering that changing requirements is a common concern in Requirements Engineering, this framework helps in identifying traces between requirements at each layer. Once the software designer identifies the requirements or design documents related to the feature to be changed, traceability helps to locate the parts of components and code that need to be maintained. By deriving systems requirements from user requirements, and by making traces from user requirements to systems requirements, one knows which user requirements triggers each system requirement.

C. Modeling user requirements using the SysML Requirements diagram

User requirements are expressed in natural language and modeled using SysML. SysML, the chosen language, is a UML profile which has the advantage of proposing one specific diagram to model requirements and their interactions. The SysML Requirements diagram can model functional and non-functional requirements, hierarchy between requirements, and other relationships such as dependency between requirements and reuse of requirements.

A SysML Requirement can also appear on other diagrams to show its relationship to design. With the SysML Requirements diagram, visualization techniques are applied from the early phases of system development. The SysML Requirements diagram is a stereotype of the UML Class diagram.

The SysML Requirements diagram allows several ways to represent requirements relationships. These include relationships for defining requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements and refining requirements. The relationships can improve the specification of systems, as they can be used to model requirements.

D. Tracing functional and non-functional requirements

The SysML Requirements diagram provides means to make traces between requirements, and between requirements and design. The trace relationship provides a general purpose relationship between a requirement and any other model element. Its semantics has no real constraints. For instance, a generic trace dependency is used to emphasize that a pair of requirements is related in a different way not defined by other SysML relationships. In Section 4, functional requirements are traced to non-functional requirements in a SysML Requirements diagram.

E. Strict relationship with architecture

The activities of evolving and maintaining software usually involves adding new requirements to already existing software. It is of utmost importance that these activities are performed considering existing software architecture. As a matter of fact, new requirements must be compatible with the software architecture used to create software products.

The software architecture is closely related to satisfying non-functional requirements, such as performance, security and safety. Using SysML Requirements diagrams, these requirements are well-documented and related to specific functional requirements. As depicted in Figure 1, requirements at each layer have relationship with the software architecture. This means that even implementation requirements have to conform with architectural decisions. Thus, each requirement will be responsible to implement and conform with requirements and constraints described in the software architecture.

F. Strict relationship of requirements with use cases

A source of problems in Requirements Engineering is the lack of traceability between requirements, and between requirements and design. The framework proposes traces between requirements and design. For instance, requirements modeled with SysML Requirements are related to UML Use Cases that are normally the starting point to software design. This relationship is described through the SysML refine relationship (Figure 2). The refine relationship provides a capability to reduce ambiguity in a requirement by relating a SysML Requirement to another model element. This relationship is typically used to refine a text-based requirement with a model. For example, how a Use Case can represent a requirement in a SysML Requirements diagram. These relationships are well-described by using SysML Requirements relationships and SysML Tables.

Figure 2. Relationship between SysML Requirements and Use Cases

SysML Requirements diagram and SysML Tables are intended to provide a bridge between traditional requirements
management specifications and other design models. For instance, SysML Requirements can be related to other SysML models, such as SysML Block diagrams, or UML Class diagrams. When combined with UML for software design, the requirements constructs provided by SysML can also fill the gap between user requirements specification, normally written in natural language, and Use Case diagrams, used as initial specification of software requirements.

IV. Application of the Framework

The framework has been applied to design a number of software-intensive systems in the field of road traffic control. Road Traffic Management Systems (RTMS) are software-intensive systems, i.e., software is an essential component and determines the overall quality and success of the system.

From the Software Engineering point of view, the development and maintenance of RTMS is a challenge due to many factors. The most important ones addressed in this article are mentioned as follows. First, requirements for RTMS are frequently changing. The major reason is that the area of road network control is still largely uncharted territory [30]. Thus, algorithms, techniques and methods are being developed by traffic engineers. In addition, policies for transportation are often being changed as well [12]. Second, legacy systems such as traffic signals control systems have to be integrated into RTMS. Due to their legacy properties, these systems are hard to adapt to be included in a new RTMS. Third, the physical network to be controlled is constantly changing, with the addition of new sensors, actuators and even extensions of the road network. Whenever the physical network changes, or a physical component of the system (for instance, a sensor such as an inductive loop) is damaged and need to be substituted, the system has to be adapted and reconfigured, which means that flexibility is a concern.

The framework has been applied to design a number of software-intensive systems in the field of road traffic control. These systems are software-intensive systems, i.e., software is an essential component and determines the overall quality and success of the system. The specific case is about the extension of an architecture (ATC) to implement road traffic control systems [26] [29]. A partial list of user requirements (11 out of 79) for this new architecture (DTCA) [27] is given as follows.

A. DTCA functional requirements

- **F1_DTCHA** - ATC should be extended to overcome its limitations.
- **F2_DTCHA** - It should be adaptive to the real traffic state thus making road traffic control less dependent on specific scenarios.
- **F3_DTCHA** - It should be able to predict the traffic state within a certain time frame enabling the proactive countering of congestion.

- **F4_DTCHA** - It should function automatically and at a lower level, thus being able to more finely apply measures (like buffering) over multiple locations in the network.
- **F5_DTCHA** - It should decentralize the control.
- **F6_DTCHA** - It should work together with ATC and existing applications.

B. DTCA non-functional requirements

A number of non-functional requirements can be derived that give guidance for the additions and choices in the technical layers. The system has to conform to and be able to provide a number of non-functional requirements described as follows.

- **NF1_DTCHA** - Flexibility: The way to do network-level control is still largely to be discovered, by simulation but also by real-life experiments. This means that control systems should allow frequent adaptations, which makes flexibility a prime requirement.
- **NF2_DTCHA** - Scalability: The scalability of multi-agent control should not be hampered by any technical aspects, thus the technical part of control systems must also be scalable.
- **NF3_DTCHA** - Timely responsiveness: One of the drawbacks of distributed systems is related to a possible poor performance. The performance problem is compounded by the fact that communication, which is essential in a distributed system, is typically quite slow. Road traffic control is a form of process control. Therefore, as mentioned above, real-time properties of applications are a major concern.
- **NF4_DTCHA** - Interoperability: When experimenting with different forms of multi-agent control, applications should be easy to connect to and operate with, which means that interoperability between road traffic control applications is important.
- **NF5_DTCHA** - Interoperability with legacy systems: The presence of legacy systems along the roads, and the circumstance that it would be too costly to ignore the existing systems, implies that interoperability with legacy systems is an important requirement.

A SysML Requirements diagram relating the functional and non-functional requirements is given in Figure 3.

C. Systems requirements

A number of systems requirements are derived from the user requirements mentioned in subsections IV.A and IV.B for each developed application. In order to illustrate the framework, only the system level requirements for the middleware module of the system are given as follows.

- **SM1**: The middleware shall offer a communication framework to make communication less primitive.
- **SM2**: The middleware shall hide implementation details of communicating processes.
- SM3: The middleware shall decouple communicating processes in time and in space.
- SM4: The middleware shall prevent blocking of processes during communication.
- SM5: The middleware shall offer mediation between communicating processes, such that they do not need to know each other.
- SM6: The middleware shall make it possible to be redundant in producers of information.
- SM7: The middleware shall make it possible the interoperability between legacy systems.

D. Relationships between requirements

Relationships can be derived from user to system requirements. For instance, the user requirement F5_DTCA is related to the system requirement SM5. Additional relationships such as this one are derived from user requirements to systems requirements. There is also relationship between system requirements and implementation requirements. For instance, requirement SM2 derives a number of implementation requirements that specifies components to be developed to fulfill this requirement. All these relationships can be expressed using SysML Tables (see, for instance, Table 1).

<table>
<thead>
<tr>
<th>Functional</th>
<th>Non-Functional</th>
<th>System requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5_DTCA</td>
<td>NF2_DTCA</td>
<td>SM3</td>
</tr>
</tbody>
</table>

V. CONCLUSION

This article proposes a framework for requirements documentation and analysis. The framework is inspired by the layered style for software architecture with the purpose of demonstrating the need to specifying requirements at different levels of detail.

The approach to model requirements at diverse layers of abstraction, and closely relating each layer to the software architecture, is useful to improve separation of concerns, and to make a strict relationship between requirements at all levels and software architecture. The advantages are the separation of concerns not only at the architectural and design levels, but also at the requirements level. In addition, knowing at which level a requirement is is useful for all stakeholders. For instance, managers are often not interested in implementation details, which have very specific requirements. On the other hand, developers can benefit from having specific requirements, not only high-level requirements that are difficult to understand and implement.

With the proposed framework it is possible to describe the relationship between requirements and between requirements and design. As a matter of fact, once a user requirement is modified, it is possible to trace forward to system-level and implementation-level requirements. Future work will focus on automating traceability between the levels of abstraction, and to facilitate automatic traceability between requirements, and between requirements and software models.

VI. ACKNOWLEDGEMENTS

This work was supported by FAPEMIG (www.fapemig.br).

REFERENCES
