Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

User requirements modeling and analysis of software-intensive systems

Michel dos Santos Soares[a,*], Jos Vrancken[b], Alexander Verbraeck[b]

[a] Universidade Federal de Uberlândia, P.O. Box 593, 38400-902 Uberlândia, Brazil
[b] Delft University of Technology, P.O. Box 5015, NL 2600 GA, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 20 July 2010
Received in revised form 7 October 2010
Accepted 14 October 2010
Available online 26 October 2010

Keywords:
Requirements
UML
SysML
Software-intensive systems

ABSTRACT

The increasing complexity of software systems makes Requirements Engineering activities both more important and more difficult. This article is about user requirements development, mainly the activities of documenting and analyzing user requirements for software-intensive systems. These are modeling activities that are useful for further Requirements Engineering activities. Current techniques for requirements modeling present a number of problems and limitations. Based on these shortcomings, a list of requirements for requirements modeling languages is proposed. The proposal of this article is to show how some extensions to SysML diagrams and tables can fulfill most of these requirements. The approach is illustrated by a list of user requirements for a Road Traffic Management System.

1. Introduction

Software-intensive systems (Wirsing et al., 2008; Tiako, 2008; Hinchey et al., 2008) are large, complex systems in which software is an essential component, interacting with other software, systems, devices, actuators, sensors and with people. Being an essential component, software influences the design, construction, deployment, and evolution of the system as a whole (ANSI/IEEE, 2000). These systems are in widespread use and their impact on society is increasing. Developments in engineering software-intensive systems have a large influence on the gains in productivity and prosperity that society has seen in recent years (Dedrick et al., 2003). Their complexity is increased due to the large number of elements and reliability factors. Thus, they must be decomposed into several smaller components in order to manage complexity and facilitate their implementation and verification. In addition, there is a need to increase the level of abstraction, hiding whenever possible unnecessary complexity, by the intense use of models. Examples of software-intensive systems can be found in many sectors, such as manufacturing plants, transportation, military, telecommunication and health care.

More specifically, the type of software-intensive systems that are investigated in this article are the Distributed Real-Time Systems. The term Real-Time System usually refers to systems with explicit timing constraints (Gomaa, 2000; Laplante, 2004). Dijkstra (2002) recognized that some applications are concurrent in nature. In concurrent problems, there is no way of predicting which system component will provide the next input, which increases design complexity. Moreover, system components, such as sensors and actuators, are often geographically distributed in a network and need to communicate according to specific timing constraints described in requirements documents.

Requirements for software are a collection of needs expressed by stakeholders respecting some constraints under which the software must operate (Pressman, 2009; Robertson and Robertson, 2006). Requirements can be classified in many ways. The first classification used in this article is related to the level of detail (the second classification is presented in Section 7.1). In this case, the two classes of requirements are user requirements and system requirements (Sommerville, 2010). User requirements are high-level abstract requirements based on end users' and other stakeholders' viewpoint. They are usually written using natural language, occasionally with the help of domain specific models such as mathematical equations, or even informal models not related to any method or language (Luisa et al., 2004). The fundamental purpose of user requirements specification is to document the needs and constraints gathered in order to later develop software based on those requirements.

Systems requirements are derived from user requirements but with a detailed description of what the system should do, and are usually modeled using formal or semi-formal methods and languages. This proposed classification allows the representation of different views for different stakeholders. This is good Software Engineering practice, as requirements should be written from different viewpoints because different stakeholders use them for various purposes.

* Corresponding author.
E-mail addresses: michel@facom.ufu.br, mics.soares@gmail.com (M.d.S. Soares).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.10.020

The process by which requirements for systems and software products are gathered, analyzed, documented and managed throughout the development life cycle is called Requirements Engineering (Sommerville, 2010). Requirements Engineering is a very influential phase in the life cycle. According to the SWEBOK (Abran et al., 2004), it concerns Software Design, Software Testing, Software Maintenance, Software Configuration Management, Software Engineering Management, Software Engineering Process, and Software Quality Knowledge Areas. Requirements Engineering is generally considered in the literature as the most critical phase within the development of software (Juristo et al., 2002; Komi-Sirviö et al., 2003; Damian et al., 2004; Minor and Armarego, 2005). Dealing with ever-changing requirements is considered the real problem of Software Engineering (Berry, 2004). Already in 1973, Boehm suggested that errors in requirements could be up to 100 times more expensive to fix than errors introduced during implementation (Boehm, 1973). According to Brooks (1987), knowing what to build, which includes requirements elicitation and technical specification, is the most difficult phase in the design of software. Lutz (1993) showed that 60% of errors in critical systems were the results of requirements errors. Studies conducted by the Standish Group (TSG, 2003) and other researchers (van Genuchten, 1991; Hofmann et al., 2001) found that the main factors for problems with software projects (cost overruns, delays, user dissatisfaction) are related to requirements issues, such as lack of user input, incomplete requirements specifications, uncontrolled requirements changing, and unclear objectives. In an empirical study with 12 companies (Hall et al., 2002), it was discovered that, out of a total of 268 development problems cited, 48% (128) were requirements problems.

Requirements Engineering can be divided into two main groups of activities (Parviainen et al., 2004): (i) requirements development, including activities such as eliciting, documenting, analyzing, and validating requirements, and (ii) requirements management, including activities related to maintenance, such as tracing and change management of requirements. This article is about user requirements development, mainly the activities of documenting and analyzing user requirements for software-intensive systems. These are modeling activities that are useful for further Requirements Engineering activities. The assumption in this article is that improving requirements modeling may have a strong impact on the quality of later requirements activities, such as requirements tracing, and in the design phase.

1.1. Research question

The main research question to be answered in this article is given as follows:

How to improve user requirements modeling and analysis for software-intensive systems?

This question is mainly answered through the early introduction of graphical models, which are used to document and analyze requirements. The identification and graphical representation of requirements relationships facilitate that traces are made. This helps in uncovering the impact that changes in requirements have in the system design. Requirements are important to determine the architecture. When designing the architecture, at least part of the functional requirements should be known. In addition, the non-functional requirements that the architecture has to conform with should be made explicit.

1.2. Article outline

Initially, a subset of a list of user requirements for a Road Traffic Management System (RTMS) is presented, using natural language, to be further modeled and analyzed (Section 2). Current techniques for requirements modeling are presented in Section 3. A number of problems and limitations related to these techniques are discussed in the same section. These shortcomings led to a list of requirements for requirements modeling languages in Section 4 and the proposed approach in Section 5 to fulfill the missing characteristics of the list. From the conclusion of Section 4, the starting point for requirements modeling languages is to use SysML diagrams and tables, which are presented in detail in Section 6. Then, SysML's constructions are extended in Section 7 and proposed to model the initial list of user requirements (Section 8). The article ends with discussion (Section 9) and conclusions (Section 10).

2. List of requirements for RTMS

The list of requirements given below is a subset from a document which contains 79 atomic requirements for RTMS (AVV, 2006). The document is a technical auditing work based on an extensive literature study and interviews, in which the stakeholders were identified. The requirements were gathered through interviews with multiple stakeholders.

The stakeholders (and the related number of requirements) were classified as: the Road Users (1), the Ministry of Transport, Public Works and Water Management (2), the Traffic Managers (10), the Traffic Management Center (8), the Task, Scenario and Operator Manager (22), the Operators (4), the Designers of the Operator's Supporting Functions (15), and the Technical Quality Managers (17). In this article the requirements of the Traffic Manager were selected as example to be modeled using SysML diagrams and constructions in Section 8. The requirements are given as follows.

Traffic Manager:

- TM4—It is expected that software systems will be increasingly more intelligent for managing the traffic-flow in a more effective and efficient manner.
- TM5—To optimize traffic flow, it is expected that gradually, region-wide traffic management methods will be introduced.
- TM6—The traffic management systems must have a convenient access to region-wide, nation-wide, or even European-wide parameters so that the traffic-flow can be managed optimally.
- TM7—It must be possible for the Traffic Managers/experts to express (strategic) "task and scenario management frames", conveniently.
- TM8—The system should effectively gather and interpret all kinds of information for the purpose of conveniently assessing the performance of the responsible companies/organizations that have carried out the construction of the related traffic systems and/or infrastructure.
- TM9—The system must support the Traffic Managers/experts so that they can express various experimental simulations and analytical models.
- TM10—The system must enable the Traffic Managers/experts to access various kinds of statistical data.
- TM11—The system must enable the Traffic Managers/experts to access different kinds of data for transient cases such as incidents.
- TM12—The system must provide means for expressing a wide range of tasks and scenarios.
- TM13—The traffic management will gradually evolve from object management towards task and scenario management.

3. Requirements modeling approaches

There are several approaches to modeling requirements. Basically, these approaches can be classified as graphics-based, purely