Towards a Framework for Specifying Software Robustness Requirements based on Patterns

Ali Shahrokni and Robert Feldt

Department of Computer Science & Engineering
Chalmers University of Technology
{nimli, robert.feldt}@chalmers.se

Abstract. [**Context and motivation**] With increasing use of software, quality attributes grow in relative importance. Robustness is a software quality attribute that has not received enough attention in requirements engineering even though it is essential, in particular for embedded and real-time systems. [**Question/Problem**] A lack of structured methods on how to specify robustness requirements generally has resulted in incomplete specification and verification of this attribute and thus potentially a lower quality. Currently, the quality of robustness specification is mainly dependent on stakeholder experience and varies wildly between companies and projects. [**Principal idea/results**] Methods targeting other non-functional properties such as safety and performance suggest that certain patterns occur in specification of requirements, regardless of project and company context. Our initial analysis with industrial partners suggests robustness requirements from different projects and contexts, if specified at all, follow the same rule. [**Contribution**] By identifying and gathering these commonalities into patterns we present a framework, *ROAST*, for specification of robustness requirements. ROAST gives clear guidelines on how to elicit and benchmark robustness requirements for software on different levels of abstraction.

1 Introduction

With software becoming more commonplace in society and with continued investments on finding better ways to produce it the maturity of both customers' requirements and our ability to fulfill them increases. Often, this leads to an increased focus on non-functional requirements and quality characteristics, such as performance, design and usability. But there is also less of a tolerance for faults and failures; by coming more reliant on software our society also increasingly requires it to be reliable and robust. *Robustness* as a software quality attribute (QA) is defined as [1]: "The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions."

The industrial project which is referred to in this paper is the development of a robust embedded software platform for running internal and third party telematics services [2]. The platform and services in this context need to be dependable in presence of erroneous input. Avoiding disturbance to a service or the platform by other services is another essential property. Considering the industrial context, robustness is interpreted as stability in presence of erroneous input and execution stability in presence of stressful environment created by external services or modules.

Robustness requirement specification is relatively unexplored in the academic literature. Much has happened in the software engineering field since the few papers that focus on robustness specification have been published [3]. However, There are common grounds between robustness and other more explored QA that can help understanding robustness better. Statements, requirements and checklists found in the literature about safety, security and dependability requirements can in some cases be applied to robustness too. This fact has been used to acquire a framework for robustness requirement (RR) specification in this paper. This commonality depends on the fact that lack of robustness is in most cases experienced and manifested as lack of other QAs or even functionality in the system. According to Lutz and Newmann [4,5], by including requirements for robustness or "defensive design" in the specifications many safety-related errors can be avoided. In this study Lutz shows that the majority of safety-related software errors in Voyager and Galileo spacecraft were interface(robustness) and functional errors.

Many failures associated with requirements are due to incompleteness [3]. This fact has motivated this study and act as a guideline to create a relevant framework that helps improving the quality of requirement specification process and ensures better completeness of the set of requirements.

In this paper we propose a framework called ROAST for specifying RR. This framework is the result of gathering data from different industrial and academic sources and applying it to the industrial project mentioned earlier. During the process a number of gaps in the existing work in the field of RR specification were identified. ROAST is the result of these steps and it can be used as a guide for RR specification and testing.

2 The ROAST framework

In this section the framework ROAST for eliciting RR and aligning specification and testing of RR is shortly described. ROAST is based on identifying patterns for specification of robustness at different abstraction levels. As mentioned earlier, robustness is not a strictly defined term and can refer to both low-level (interface and input validation, failure handling) and high-level (service degradation, availability, reliability and dependability) requirement types.

There are three main ideas behind the method: (a) specification levels, (b) requirement patterns, and (c) alignment from requirements to testing. The first 2 parts are shortly described in this paper and the alignment from requirements to testing will be discussed in future publications.

Like many NFRs, RR are often summative in nature. This means that they specify general attributes of the developed system and not specific attributes for specific, 'local' situations. For example, while a functional requirement (FR) for a certain feature of a telematics system ('system should support being updated with applications during runtime') can be judged by considering if that specific feature is present or not, a RR ('system should be stable at all times, it cannot shut down because of erroneous inputs or components') requires testing of a large number of different system executions. So while a FR talks about one specific situation, or a definite sub-set of situations, a RR summarizes aspects of the expected system behavior for a multitude of situations.

To make RRs testable they need to be refined into specific behaviors that should (positive) or should never happen (negative). Early in the development of a software system users or developers may not be able to provide all details needed to pinpoint such a behavior (or non-behavior). However, it would be a mistake not to capture more general RRs. Our method thus describes different information items in a full specification of a robustness behavior and describes different levels in detailing them. This is similar to the Performance Refinement and Evolution Model (PREM) as described by [6,7], but specific to robustness instead of performance requirements. The different levels can be used to judge the maturity of specifying a requirement or as a specific goal to strive for.

Since RR are often summative, i.e. valid for multiple different system situations, they are also more likely, than specific functional requirements, to be similar for different systems. We can exploit this similarity to make our method both more effective (help achieve a higher quality) and efficient (help lower costs). By creating a library of common specification patterns for robustness, industrial practitioners can start their work from that library. Thus they need not develop the requirements from scratch and can use the pattern to guide the writing of a specific requirement. This can both increase quality and decrease time in developing the requirements. Our approach and the pattern template we use is based on the requirements patterns for embedded systems developed by Konrad and Cheng, that are in turn based on the design patterns book [8,9].

The verification of different robustness behaviors should be aligned with the RR. Based on the level of requirement and the pattern the requirement is based on different verification methods are applicable and relevant. We make these links explicit in order to simplify the verification and testing process. Note that the verification method that is relevant for a certain pattern at a certain level may not actually be a testing pattern. For pattern levels that are not quantifiable or testable, the verification method may be a checklist or similar technique.

Figure 1 gives an overview of the method we propose, and shows both the levels, robustness areas with patterns and attached verification methods. In the following we describe each part in more detail.

2.1 Robustness Requirements Levels

To have a similar model for RR as presented in [7], we need to identify the factors that can affect robustness. These factors are generally not the same as the ones affecting performance. We also need to specify these factors in more detail for them to be useful for practitioners. Since these factors might be simulated at many different levels of fidelity it is important to realize that level 2 is rather a continuum of lower or higher fidelity approximations of level 3. We also use our patterns to clarify different ways for quantifying qualitative requirements.One important such quantification is transforming a qualitative requirement for the system as a whole into a local variant specific to certain components and/or sub-systems. We have introduced named levels for these two different types of qualitative requirement types.

Figure 1 shows these different levels of a requirement in a diagram. The two main axis of this model are ‘Specificity’ and ‘Realism’. The former increases when we detail the scope of the requirement, from a global, system-wide requirement to be localized to a component or sub-system. It also increases when we quantify how and to what degree the requirement is to be fulfilled. The ‘Realism’ axis increases when we mimic the realism of the factors that affects system execution and its robustness. When we do no or little specification of these factors the requirement is a RR-3 requirement, i.e. specific but not realistically specified. As we describe the factors more realistically we increase realism into a RR-4. We can strive to reach RR-5 by using real-world values and workloads in describing the factors.

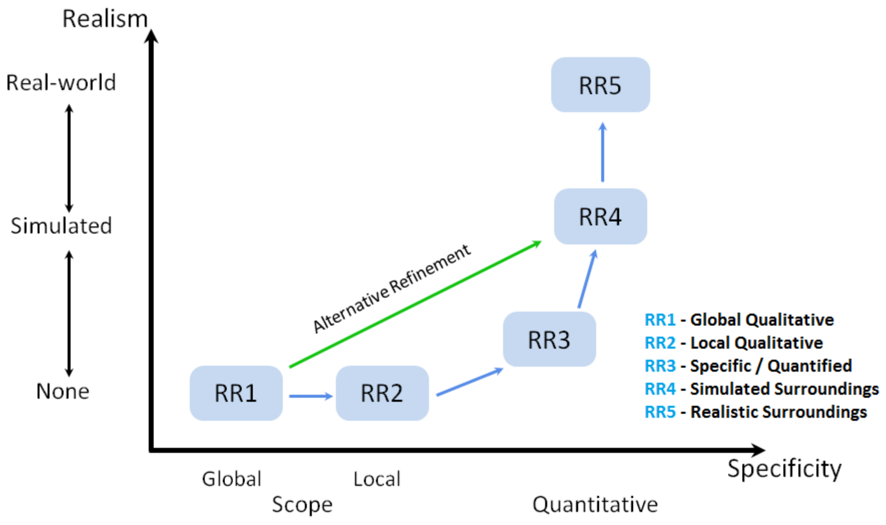

Fig. 1. Robustness Requirements Levels and Typical Refinement

2.2 Robustness Specification Patterns

Konrad and Cheng introduced requirements patterns for embedded systems [8]. They modify the original template introduced by Gamma et al by adding information items for ‘Constraints’, ‘Behavior’ and ‘Design Patterns’ and by deleting the 'Implementation' and 'Sample Code' items. The focus for Konrad and Cheng is to specify the connection from requirements to the design as specified in UML diagrams. Even though this can be a worthy goal also for our method in the long-term we would rather keep design alternatives out of the robustness patterns. Primarily because multiple different designs will be able to support RR; pre-specifying the allowed solutions would be too restrictive for developers. The connection from the patterns to verification and test methods that we propose is more natural since each requirement will typically need to be tested and each verification activity should be motivated by some requirement. We have thus modified the template by Konrad and Cheng to reflect this difference in purpose. A pattern captures a whole family of related requirements but that can vary according to our levels.

Some of the patterns in ROAST are similar to the ones introduced by Lutz [4] which account for the majority of safety-related errors Galileo and Voyager However, when working on a platform in the presence of many services, with little runtime control, it is essential to predict not only how the system can be affected through interfaces but even how it behaves when sharing resources with other services. The 14 identified robustness patterns are presented in table 1 where *IS* is Input Stability, *ES* Execution Stability and *M* stands for Means to achieve robustness:

Table 1. Robustness Specification Patterns

N	**Pattern**	**Category**
1	Specified response to out-of-range and invalid inputs	IS
2	Specified response to timeout and latency	IS
3	Specified Response to input with unexpected timing	IS
4	High input frequency	IS
5	Lost events	IS
6	High output frequency	IS
7	Input before or during startup, after or during shut down	IS
8	Error recovery delays	IS
9	Graceful degradation	M
10	All modes and modules reachable	M
11	run-time memory access in presence of other modules and services	ES
12	Processor access in presence of other modules and services	ES
13	Persistent memory access in presence of other modules and services	ES
14	Network access in presence of other modules and services	ES

The patterns presented in this section are partly elicited by studying earlier requirement documents from similar projects and partly through expertise provided by the participants in the project who are mainly experienced people in the field of requirement engineering. Earlier academic work presented above helped us complete and reformulate already identified patterns.

3 Conclusion

The state of the art and practice concerning robustness requirements and testing is rather immature compared to that of other quality attributes. The proposed framework, ROAST, is a unified framework for how to interpret robustness and specify and verify robustness requirements.

ROAST follows a requirement as it often evolves from a high level requirement to a set of verifiable and concrete requirements. Therefore ROAST consists of different levels of specification that follow the most typical requirement specification phases practiced in the industry. As presented in ROAST, requirements engineering process tends to start from high level requirements and break them down into more specific and measurable ones. Therefore, ROAST can be incorporated into the activities of most companies with minimal change to the rest of the process. The commonality often seen between robustness requirements in different projects is captured in patterns. For different patterns and levels different verification methods will be more or less useful.

Initial evaluation of ROAST has been carried out in an industrial setting. Preliminary results are promising and show that the resulting requirements are more complete and more likely to be verifiable. Further evaluation is underway.

References

1. IEEE Computer Society, "Ieee standard glossary of software engineering terminology," IEEE, Tech. Rep. Std. 610.12-1990, 1990.
2. A. Shahrokni, R. Feldt, F. Petterson, and A. Back, "Robustness verification challenges in automotive telematics software," in *SEKE*, 2009, pp. 460–465.
3. M. Jaffe and N. Leveson, "Completeness, robustness, and safety in real-time software requirements specification," in *Proceedings of the 11th international conference on Software engineering.* ACM New York, NY, USA, 1989, pp. 302–311.
4. R. R. Lutz, "Targeting safety-related errors during software requirements analysis," *Journal of Systems and Software*, vol. 34, no. 3, pp. 223 – 230, 1996.
5. P. Newmann, "The computer-related risk of the year: weak links and correlated events," in *Computer Assurance, 1991. COMPASS '91, Systems Integrity, Software Safety and Process Security. Proceedings of the Sixth Annual Conference on*, Jun 1991, pp. 5–8.
6. C.-W. Ho, M. Johnson, and L. W. E. Maximilien, "On agile performance requirements specification and testing," in *Agile Conference 2006.* IEEE, 2006, pp. 46–52.
7. C.-W. Ho, "Performance requirements improvement with an evolutionary model," PhD in Software Engineering, North Carolina State University, 2008.
8. S. Konrad and B. H. C. Cheng, "Requirements patterns for embedded systems," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE02)*, Essen, Germany, September 2002.
9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns.* Boston, MA: Addison-Wesley, January 1995.