On Exceptions, Exception Handling, Requirements and Software Lifecycle

Alexander Romanovsky
Newcastle University, UK
alexander.romanovsky@ncl.ac.uk

Abstract

This is a position statement accompanying the HASE-2008 panel on exception handling.

1. Introduction

It is often the case that faults and fault tolerance are not dealt with left until late implementation phases [1]. In a similar way, exceptions and exception handling are typically viewed as language features. Fortunately, it is becoming clear now that exception handling should be an immanent part of all development phases [2]. It is difficult to underestimate the importance of identifying the correct and complete set of requirements for exceptions and exception handling. Thus, at a meeting in Imperial College, London in 2005 D. Parnas claimed that up to 80% of requirements may have to deal with exceptions and emphasized that there *is no practical upper bound on the number of things that can go wrong* [3].

There is no widely accepted methodology for eliciting these requirements but this is now clearly becoming an area of very active research with several groups already contributing: paper [4] extends use cases to include a description of exceptional behaviour which uses sequences of actions performed by the system; paper [5] does the same to express the situations that can prevent the system from achieving its goals; an approach in [6] is based on usage models to allow specification and modeling of exception handling using a requirement state machine language; and paper [7] discusses a semi-formal specification of fault-tolerance requirements using the concept of deviation from requirements.

2. Issues

Several issues still need to be clarified and addressed to make the elicitation of exception handling requirements a well-established practice.

Issue 1. What are exceptions and what is exception handling at the requirement level? These should not be confused with the implementation level mechanisms for forward error recovery. It is important to understand what is normal and what is abnormal for requirements. These concepts should be directly related to the concept of fault assumptions and the system approach, perused considering the system and its environment as potential sources of exceptions.

Issue 2. Why do we need these requirements? Why not use standard features to express normal and exceptional requirements? What are the important differences and interplays between them which need to be captured?

Issue 3. What are we not doing right at the moment? The existing solutions do not offer stepwise guidelines to eliciting the requirements in a systematic and rigorous way, so that all stakeholders' concerns are taken into account, inconsistencies and incompleteness avoided, and traceable requirements which can be used at the later phases produced.

Issue 4. What do we do with these requirements? How can they be smoothly transformed and used in the later phases (e.g. architectural design, specification)?

3. References

[1] A. Romanovsky. A looming fault tolerance software crisis? *ACM SIGSOFT SE Notes*. 32(2). 2007.

[2] R. de Lemos, and A. Romanovsky. Exception handling in the software lifecycle. *Comp Syst Sci&Eng,* 16(2). 2001.

[3] Requirements Quarterly. Newsletters of the RE Special Group of the British Computer Society. RQ36, June 2005.

[4] C. M. F. Rubira, R. de Lemos, G. R. M. Ferreira, and F. Castor Filho. Exception handling in the development of dependable component-based systems. *Software – Practice and Experience*. 35. 2005.

[5] A. Shui, S. Mustafiz, and J. Kienzle. Exception-Aware Requirements Elicitation with Use Cases. *Advanced Topics in Exception Handling Techniques*. LNCS-4119. 2006.

[6] W. Bail. An Approach to Defining Requirements for Exceptions. *Ibid.*

[7] A. Berlizev, and N. Guelfi. Engineering Fault-tolerance Requirements using Deviations and the FIDJI Methodology. *Workshop on Methods, Models and Tools for Fault Tolerance*, IFM-2007. Newcastle Un., CS-1032. UK. 2007.