Guest Editors’ Introduction to the Special Section on Exception Handling: From Requirements to Software Maintenance

Alessandro Garcia, Alexander Romanovsky, and Valérie Issarny

1 INTRODUCTION

Since their first appearance in the 1970s, exception handling models in mainstream programming languages, frameworks, and platforms have not changed much. Modern exception handling mechanisms obviously differ in how to specify exceptions and their handlers, how to declare exceptions in module interfaces, how to specify protected regions, and how to continue the control flow after exceptions have been handled. However, these differences are insignificant, and exception handling models still rely on the same well-known traditional principles for structuring error recovery in software systems, such as supporting explicit separation between normal and exceptional behaviors.

Moreover, exception handling is still an afterthought issue and not much support for it has been provided throughout the software process, from requirement elicitation to software maintenance and evolution. With the heterogeneity and complexity of contemporary software applications growing, there is a pressing need for an in-depth reflection about the most appropriate mechanisms and methods for exception handling across the software lifecycle. Sometimes, given the increasing uncertainty in open software systems, it is even necessary to identify and apply more dynamic mechanisms for exception handling. In fact, modern software applications, such as context-aware mobile systems and service-based processes, pose new error handling challenges, partly because they cannot be designed as a coherent whole. Their decentralized character means that collaborative exception handling is potentially more difficult to develop correctly and consistently. Separate ownership of components means that we cannot design for central control over (exception handling) evolution and upgrades.

Ten years ago, in the beginning of the new century, these concerns were first explicitly debated in the first special issue devoted to exception handling in the IEEE Transactions on Software Engineering. In many ways, this issue set the research agenda for the coming years, stimulating the research community’s growing interest in issues related to exception handling engineering. Unfortunately, the take-up of the research ideas has not been very successful and a statement made in the issue still rings true: “many failures are caused by incomplete or incorrect handling of these abnormal situations.” Even though such an inaugural special issue was of primary importance to the field, it was clear that we still did not know much about the real benefits and drawbacks of conventional exception handling in real-life software projects. It became clear that, in the first place, we needed a better understanding of why conventional exception handling models were not working properly in mainstream software development.

Over the last decade there have been some advances in this direction. A number of interesting empirical studies have emerged, providing some initial answers. Some recent field studies have determined that error-handling design in industrial applications typically exhibits poor quality independently of the underlying programming language or the application domain. Also, several researchers have recently pointed out that exceptions are a global design issue and should be treated systematically across the software lifecycle. However, many burning questions have not yet been answered. We still need to understand why conventional mechanisms for coping with exceptions continue to remain one of the major sources of poor software quality and failures in real-life software projects.

The concern to improve error-handling quality has inspired research on exception handling across different communities over the last decade. In the software engineering field, the interest in it has been steadily growing in recent years. For instance, with the new software development techniques (such as context-oriented programming, model-driven development, and aspect-oriented programming) about to achieve a certain level of maturity, it is vital for software engineering researchers and practitioners to understand their contribution to improving exception-handling design. In fact, in the past decade some authors have investigated the extent to which such contemporary software techniques can improve the modularity and robustness of
error handling. Another area of growing interest is ensuring that the dependability requirements are rigorously and traceably addressed in developing critical systems by using exception-handling concepts. Many researchers and practitioners realize now that these concepts provide beneficial and inductive ways of representing the system structure, including requirements, design models, and code. Today, the relevance of exception handling becomes even more evident when we look at the amount of recent work on exception handling in key software engineering areas, such as modeling languages and model transformations, software evolution and refactoring, empirical studies, software product lines, ubiquitous computing, and formal methods.

2 About This Special Section

The purpose of the special section is to gather together a collection of papers that present innovative, well-validated ideas on how to cope with the growing complexity of exception handling. The section is intended to serve as a key reference point for researchers, practitioners, and educators, helping them to understand the latest innovations, trends, experiences, and concerns involving exception-handling issues in software engineering. In response to our call for papers, we received 34 submissions. Each paper was reviewed by at least three expert referees. After two rounds of the peer-review process, we selected four papers that present new findings on the interplay of exception handling and software engineering.

The first paper of this special section introduces important lessons learned about the human dimension in tasks related to exception handling in software systems. In the paper “Understanding Exception Handling: Viewpoints of Novices and Experts,” Hina Shah, Carsten Görg, and Mary Jean Harrold report user studies conducted with programming novices and experts. Their findings indicate that novice developers tend to ignore exceptions when developing programs, use exception mechanisms primarily for debugging the code, and dislike the forced handling of exceptions that Java imposes. On the other hand, the expert developers usually exploit exception handling to convey important information to the user, and train themselves to reason about both normal and abnormal behavior simultaneously. The authors also report the policies that experts use to systematically handle exceptions.

In “Exception Handling Patterns for Process Modeling,” Barbara Lerner, Stefan Christov, Leon Osterweil, Reda Bendraou, Udo Kannengiesser, and Alexander Wise argue that a more complete analysis of a software process requires the inclusion of details about how to react in the presence of exceptional situations. They provide a catalog of general patterns capturing the relationship between exception handling tasks and the normative process. These patterns were observed over many years of process modeling in different domains, such as business, negotiation, and healthcare. The authors also discuss the advantages and drawbacks of three notations—UML 2.0 Activity Diagrams, BPMN, and Little-JIL—to represent the exception handling patterns.

Certain contemporary application domains require specialized exception handling models. The next paper, “A Framework for Programming Robust Context-Aware Applications” by Devdatta Kulkarni and Anand Tripathi, presents a novel exception handling mechanism for designing and implementing robust context-aware applications. Context-awareness is a central aspect of pervasive computing applications, characterizing their ability to adapt and perform tasks based on the ambient context conditions. The proposed mechanism enables the incorporation of programmed recovery actions for several error categories, such as errors in context-based reconfigurations, service-level operational exceptions, and context invalidations, into mobile applications. Exception handlers can be attached not only to objects, but also to specific “roles” in the system. The error handling possibilities also include collaborative recovery actions involving members playing different roles.

The last paper of our special section, entitled “Exception-Handling for Repair in Service-Based Processes,” is written by Maria Grazia Fugini, Barbara Pernici, Gerhard Friedrich, Gaston Tagni, and Enrico Mussi. A fundamental goal of service-based processes is to achieve the correct results of the service composition upon exceptional conditions, repairing the process during its execution. The authors state that conventional exception handling approaches for repairing service-based processes are limited in many ways. First, the required definition of all exception handlers at design time imposes a high development effort. Second, it makes it difficult to take into consideration the specific causes of process exceptions. Third, exception handling usually requires the reexecution of (the parts of) the process, which often results in unnecessary service re-invocations. The paper describes a novel model-based approach to exception handling in service-based processes that takes into account the fault origins, the structure of the process, and the process execution state. The proposal is based on the concept of self-healing systems, namely systems that are able to recognize exceptions and to proactively react to faults by setting proper repair plans.

Acknowledgments

We would like to thank the authors of all of the submitted papers for their interest in our special section. The support of the former IEEE Transactions on Software Engineering Editor-in-Chief Jeff Kramer and the IEEE Computer Society staff has helped us a lot in the preparation of this section. Finally, we are grateful to all reviewers whose time and efforts have made this section possible. More than 120 experts on exception handling and other software engineering subfields contributed to the success of this special section.

Alessandro Garcia
Alexander Romanovsky
Valérie Issarny
Guest Editors
Alessandro Garcia has been an assistant professor in the Informatics Department at PUC-Rio, Brazil, since January 2009. He was a Lecturer in the Computing Department at Lancaster University, United Kingdom, from 2005 to 2009. He was previously a software architect at the IBM Almaden Research Center and a visiting researcher at the University of Waterloo, Canada. His main research interests are in the areas of exception handling, advanced modular programming, software architecture, software product lines, and empirical software engineering. He has been serving as a committee member of premier international conferences on software engineering, such as ICSE, AOSD, SPLC, OOPSLA/SPLASH, and AAMAS. He received many awards and distinctions, including the Best Dissertation Award (Computer Brazilian Society, 2000), Best Researcher Award (Lancaster University, 2006), Distinguished Young Scholar (PUC-Rio, 2009), Research Productivity Grant (CNPq, 2009), and Young Scientist Fellowship (FAPERJ). He has been involved in several projects, funded by EC and Brazilian research agencies, and his research contributions have been exploited by a number of industry partners, such as Siemens, SAP, IBM, and Motorola.

Alexander Romanovsky is a professor at the Centre for Software Reliability, Newcastle University, United Kingdom. His main research areas are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring, and verification of fault tolerance. He has been involved in a series of EC and UK projects focusing on various aspects of dependability, including PDCS, DeVa, DSoS, and CABERNET funded by the EC, and DISCS and DOTS sponsored by EPSRC/UK. He was the coordinator of the Rigorous Open Development Environment for Complex Systems (RODIN) ICT STREP Project (2004-2007). He is now the coordinator of a major FP7 DEPLOY Integrated Project, involving more than 60 researchers from 13 partners, including Siemens, Bosch, SAP, and other large industrial companies, concerned with cost-effective, rigorous development of dependable software systems and services. Professor Romanovsky has more than 200 publications in the area of dependable computing. He was the coeditor of the IEEE Transactions on Software Engineering special issue on current trends in exception handling published in 2000.

Valérie Issarny is Directrice de recherche at INRIA, France. Since 2002, she has been the head of the INRIA ARLES research project-team at the Paris-Rocquencourt research center. Her research interests relate to dependability, distributed systems, software engineering, middleware, and pervasive computing. She has been involved in a number of European and French projects and initiatives in the above areas. She has further been a member of organization and program committees of leading events in these areas. She was the program committee cochair of the Middleware '08 conference and ESEC/FSE '09. She is also an associate editor of the ACM Computing Surveys.