Foreword

For years exception handling has been a side issue in language and system design. I could not be more pleased to see the recent surge of interest in this topic, as reflected in recent special issues of Transactions on Software Engineering, the workshop yielding the papers in this volume, and future planned workshops. Moreover, in reading through these papers, I was surprised at the breadth of topics covered. Programming language issues are no longer the central focus; researchers are looking into broader topics such as workflow processes, distributed systems, system design, etc.

This broad interest is understandable. Although exception handling originated with sequential programming languages, today's distributed, component-oriented computing world raises new language design and implementation issues as well as new concerns about how to deal with exceptions in system design. I am glad to see such vigorous exploration of these issues.

Several of the papers deal with system design, but there are still issues that remain to be explored. For example, I have long (but quietly) advocated dealing with exception handling issues early in the design of a system. Unfortunately, there is a natural tendency to focus on the main functional flow of a system, ignoring the impact of exceptional situations until later. Design guidelines for using exceptions are lacking; indeed, there is little appreciation of the *need* for such guidelines. For example, even leaving aside distributed and concurrent programming issues, what design guidelines exist to help decide which preconditions should be checked inside a module and which should be the caller's responsibility? For which exception situations should a module provide a Boolean function such as At_End_of_File so a programmer can test for the existence of the situation without calling the method that raises the exception? When an exception occurs, what additional information should be provided to the handler (even at the cost of a breach in information hiding), and what system-wide conventions define what information will be provided and how it will be provided?

Another topic that concerns me is the interaction between exceptions and optimization. Long ago, a PL/I implementer observed that the existence of exceptions raised by language-defined operations could kill many optimizations. For example, if code is reordered to optimize usage of a pipelined floating point processor, some textually-later operations may be performed in parallel with the floating point computations. Suppose a complex assignment that occurs textually later in the code is actually started while waiting for the floating point processor to finish. If an overflow exception is raised after the assignment is completed, the handler may not expect the variable to be updated. If the assignment is interrupted, the variable may be left in a partially updated state. When a language permits such reorderings, then the state seen at an exception handler is less defined than a programmer might think. When a language does not permit such reordering, then code may run slower. Deciding what computational states are allowed after occurrence of an exception raised by a language-defined operation is a tricky proposition; it occupied a lot of discussion during the Ada design.

I raise these points only to suggest that despite the swell of papers and work on exception handling, there are still technical nuggets to be mined by the adventurous explorer. The nuggets that can be found in this book are still only the beginning.

January 2001

John B. Goodenough
Carnegie Mellon University

Preface

Modern software systems are becoming more complex in many ways (including their size, modules, components, distribution, concurrence, interoperability) and have to cope with a growing number of abnormal situations which, in their turn, are increasingly more complex to handle. The most general way of dealing with these problems is by incorporating exception handling techniques in software design. In the past, various exception handling models and techniques have been proposed, many of which are part of practical languages (e.g. PL/I, Ada, CLU, LISP, Smalltalk, Eiffel, BETA, C++, Java) and software composition technologies (e.g. CORBA, COM, or Enterprise Java Beans).

In spite of these developments and a better understanding of exception handling techniques by today's software designers, many problems still persist because the mechanisms embedded in exception handling systems and the methodologies of using them are often error-prone due to the complexity of exception code. Moreover, specification, analysis, verification, and testing of programs and systems intended for coping with exceptional situations are far from being straightforward. Finally, very often the exception handling features used are not compatible with the programming language features or the methodology in which they are employed, thus creating a gap that can cause mistakes or unnecessary complications. Developing new exception handling techniques should go together with developing advanced models, languages, and paradigms and take into account various specific application domains as well as users' experience of employing the existing ones.

In the early 1970s John Goodenough[1] was the first to define the seminal concepts for exception handling models, techniques, and applications. He defined exception conditions and handling as follows: "Of the conditions detected while attempting to perform some operation, *exception conditions* are those brought to the attention of the operation invoker. ... Bringing an exception condition to invoker's attention is called *raising* an exception. The invoker's response is called *handling* the exception." Knudsen[2] defined exception conditions in a more general sense: "an exception occurrence is a computational state that requires an extraordinary computation". This book brings together a collection of 17 papers addressing a wide range of issues in exception handling techniques, reflecting this broad view and the importance of exception handling in today's software systems.

This collection aims at discussing important topics of developing advanced exception handling techniques and of applying them in building robust and dependable systems. The papers presented describe well-established results, recent developments on

---

1 Goodenough, J.B.: Exception Handling, Issues and a Proposed Notation. Communications of ACM, **18**, 12 (1975) 683-696

2 Knudsen. J.L.: Better Exception Handling in Block-structured Systems, IEEE Software, May (1987) 40-49

specific issues, and practical experience. The research problems addressed include linguistic issues and programming constructs, integration of exception handling models with object-oriented programming principles, and methodologies for employing exception handling in software designs incorporating concurrent and distributed components. The book offers an exposition of techniques and experiences concerning exception handling in mission-critical systems, databases, and workflow process management systems.

This book is composed of five parts, which deal with topics related to exception handling in the context of programming language models, design methodologies, concurrent and distributed systems, applications and experiences, and large-scale systems such as databases and workflow process management systems.

The first part focuses on language support for exception handling. It comprises three papers which describe how exception handling systems are integrated into statically and dynamically typed programming languages. These systems differ in how exceptions are represented; what kind of control structures are provided to signal and handle exceptions; the semantics of these control structures and its effect on the expressive power and the program quality and, finally, how the exception handling primitives can be implemented. The paper by Jørgen Lindskov Knudsen presents a new design for augmenting the static exception handling model in BETA by a dynamic exception handling model, and discusses how these two models can coexist and interact. Finally, the strengths of both models are discussed. Christophe Dony describes the design and implementation of a fully object-oriented exception handling system for Smalltalk which incorporates class handlers. And finally, Kent M. Pitman presents the rationale and evolution of exception handling models in the LISP language family.

The second part deals with the design and modeling of exception handling structures. So far, there have not been enough studies which consider the most effective ways of using the existing systems, their limitations, and how to overcome them. Bjarne Stroustrup discusses the effective and practical use of exception handling libraries in C++. The paper by Yolande Ahronovitz and Marianne Huchard is concerned with the design of exception class hierarchies and their connection with predefined exception hierarchies. Finally, the paper by Anna Mikhailova and Alexander Romanovsky describes the evolution of application programs and, in particular, the issues raised by the integration of new exceptions in module interfaces, and proposes some effective solutions.

The papers in the third part focus on exception handling issues in concurrent and distributed computing. With cooperative concurrency, an exception encountered by one component may require actions by the components cooperating with it. Also, exceptions concurrently raised by different components need to be resolved together to determine the global handling action to be undertaken. The paper by Valérie Issarny presents a programming language level support addressing these problems. Several particular problems arise when the cooperating autonomous components of an application are distributed. Thus, in agent-based systems components can migrate in the network. Anand Tripathi and Robert Miller propose a framework for exception

handling in agent-based systems. Many concurrent and distributed systems are structured using shared objects and atomic actions. Such systems support both competitive and cooperative models of concurrency. The exception handling approaches based on action-oriented structuring of such concurrent systems are outlined in the survey paper by Alexander Romanovsky and Jörg Kienzle. Finally, Marta Patiño-Martinez, Ricardo Jiménez-Periz, and Sergio Arévalo discuss exception handling mechanisms developed for replicated object groups, employing the transactional model of atomic actions.

The papers in the fourth part of the book focus on practical problems and experience of integrating exception handling techniques in real-world systems. The topics addressed include techniques for building dependable systems, integration of distributed components using COM, and implementation mechanisms for checkpointing the state of partially executed programs for reliability or mobile computing. Integration of exception handling in software system design is essential for building mission-critical and dependable systems. The paper by Charles Howell and Gary Vecellio presents various patterns of exception handling derived from building mission-critical systems using Ada. Alessandro F. Garcia and Cecília M.F. Rubira propose a reflective architecture for systematically integrating exception handling mechanisms at various stages of designing an object-oriented dependable system. The paper by Bjørn Eigil Hansen and Henrik Fredholm presents techniques for adapting the C++ exception handling model to the COM exception model. And, finally, Tatsurou Sekiguchi, Takahiro Sakamoto, and Akinori Yonezawa show how an exception handling system can be used to develop a portable implementation of control operators which allows the manipulation of program continuations in imperative languages.

The last part is concerned with exception handling techniques for information systems, with a special focus on database and workflow management systems. The paper by Elisa Bertino, Giovanna Guerrini, and Isabella Merlo discusses the issues related to exceptions in object-oriented databases. These exceptions can arise in two situations: when the data not conforming to the prescribed schema are stored in the database, and when the abnormal conditions occur during data processing. Workflow process management systems support the modeling and enactment of enterprise-wide processes. Such processes can sometimes encounter exceptional conditions during their execution due to errors in the modeling of business activities, mistakes made by the people involved in a process, or failures in the underlying computing infrastructure. Fabio Casati and Gianpaolo Cugola discuss how such exceptions and failures occurring in business process applications, which model and mimic human group activities, can be classified and handled by higher-level dedicated exception handling systems. The paper by Dickson K.W. Chiu, Qing Li, and Kamalakar Karlapalem presents an overview of a workflow management system integrating several original approaches to exception handling: the automated and cooperative resolution of expected exceptions, handling via workflow evolution, and developing support for user-driven handling of unexpected exceptions.

The starting point of our work on this book was the workshop on Exception Handling in Object-Oriented Systems that we organized at ECOOP 2000[3]. Later on we decided to widen the scope of the book and, after inviting several workshop participants to prepare chapters for this book, extended this invitation to a number of other leading researchers working on different aspects of exception handling. It is only natural that the choice of contributors to this book reflects our personal views on this research area. However, we are hopeful that reading this volume will prove rewarding to all computer scientists and practitioners who realise the importance of dealing with abnormal events in building robust and safe systems.

Our thanks go to all authors, whose work made this book possible. Many of them also helped during the review process. We also would like to thank Prof. John Goodenough for contributing the foreword to this book. Finally, we would like to thank Alfred Hofmann of Springer-Verlag for recognising the importance of the project and publishing this book.

January 2001

Alexander Romanovsky
Christophe Dony
Jørgen Lindskov Knudsen
Anand Tripathi

[3] Romanovsky, A., Dony, C., Knudsen, J.L., and Tripathi, A.: Exception Handling in Object-Oriented Systems. In Malenfant, J., Moisan, S., and Moreira, A., (eds.): Object-Oriented Technology. ECOOP 2000 Workshop Reader. Lecture Notes in Computer Science Vol. 1964. Springer-Verlag, Berlin (2000) 16-31