Lecture Notes in Computer Science 1009
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Manfred Broy Stefan Jähnichen (Eds.)

KORSO: Methods, Languages, and Tools for the Construction of Correct Software

Final Report

Springer

Volume Editors

Manfred Broy
Institut für Informatik, Technische Universität München
Arcisstr. 21, D-80333 München, Germany

Stefan Jähnichen
Fachbereich 13 - Informatik, Technische Universität Berlin
Franklinstr. 28-29, D-10587 Berlin, Germany

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

KORSO: methods, languages, and tools for the construction of correct software : final report / Manfred Broy ; Stefan Jähnichen (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 1995
(Lecture notes in computer science ; 1009)
ISBN 3-540-60589-4
NE: Broy, Manfred [Hrsg.]; GT

CR Subject Classification (1991): D.2, D.3, F.3

ISBN 3-540-60589-4 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10487181 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

Correctness is a decisive quality attribute of software systems. Although technically used for the consistency of a software system with its formalized requirements in a more pragmatic usage correctness comprises the adequacy of the formalized requirements and the consistency of the software with them. Formal methods provide a rigorous solution to the difficult problem of achieving and demonstrating correctness. Since exhaustive consistency tests are not practicable for large software systems, formal methods are often a prerequisite for the use of risky applications.

The papers published in the present volume are a representative selection of the results of work carried out in the project KORSO ("Korrekte Software"). This project was funded by the German Federal Ministry for Research and Technology. As early as 1990, the ministry recognized the need for a substantial improvement in the quality of software products and adopted this as a research topic of clear national interest. Two integrated joint projects were subsequently initiated, one, REMO, dealing with the evaluation of tools and methods for attaining high security standards for the use of computer systems, and a second, KORSO, concerned with the development of tools and methods for improving the quality of software products.

This second project was quite consciously entrusted to partners with different knowledge profiles and technical orientations, the idea being that the project should produce as its result a status report on the relevant methods available, and – more importantly – a demonstration of their practical implementability. This is the reason why the scientific partners in the project greatly outweighed the industrial partners in terms of numbers. The project worked for a period of three years as a coordinated group of individual partners with individual objectives but also with close cooperation on specific research topics.

The primary aim of the KORSO project was to perfect and also to test the theoretical foundations for improving software quality, and to implement already known techniques for applications of practical relevance. Its goal was not to develop marketable products for ensuring the quality of software – given the state of the art at the time, that would have been a rather bold undertaking. KORSO must therefore be seen as an evolutionary prototype-oriented project, laying the ground for a systematic, quality-driven software engineering of the future. The evolutionary nature of the project is based primarily on the insight that new concepts can only be introduced if, at the same time, traditional, proven methods are retained, and the new concepts integrated with these.

The work done in the KORSO project includes the conception of a general methodological framework for software production; the definition of a generic description and modelling language; support for deductive correctness-preserving development steps using mathematically-based calculi and tools based on them; and, of course, a constant critical review of all the work in progress by means of numerous accompanying case studies.

The present volume follows in its structure the division of the overall re-

search effort into different research topics. It contains a part of papers dealing with methodological approaches to the development of correct software (methods), and a part for the used formal specification and implementation languages. The systems and logical frameworks part gives new concepts for development systems, whereas the tool part evaluates some experimental tools used in the project. The last part of the volume is devoted to case studies carried out in the KORSO-project. The papers are by no means homogeneous, reflecting as they do the different expectations of the partners; instead, they frequently describe the specific results and perspectives of the various partners.

To help readers in selecting the papers that are of interest for them, an overview of the different papers is given below. In addition the first paper provides an introduction to the subject. It is concerned thematically with methods, but describes an accepted approach to the development of correct software on the basis of axiomatic specifications. The initial sections of the paper provide a motivation for the use of formal methods. The paper does, however, also present a brief overview of the current state of the art and specifies the research goals in this area.

The first part is devoted to *methods*. While the introductory paper, **Correct Software: From Experiments to Applications**, focuses on methodological questions, here the individual approaches are looked at in detail and illustrated by means of small examples. The first papers in this area present different process models with different aims.

The first of these, **A Method for the Development of Correct Software**, follows on directly from the introductory papers, describing a method based on axiomatic specifications. The model of a development graph and its handling and manipulation are described in technical terms. This method is applied in the next paper **Realizing Sets by Hash Tables** which looks at a special refinement problem, and demonstrates the solution with the KORSO development graph.

A different approach is described in the paper **Event Automata as a Generic Model of Reactive Systems**, which gives a mathematical framework with various instantiations for concurrency semantics or object-oriented methods. This is a solid mathematical base for object-oriented methods, as described in **On Object-Oriented Design and Verification**, which also focuses on the important point of verification.

Another view of method support is given in the paper **Design of Modular Software Systems with Reuse** which looks specifically at the concept of reuse, describing a process model in which existing and implemented modules can be integrated as early as the specification phase. The last paper of this part, **AVL Trees Revisited: A Case Study in SPECTRUM** shows a pass through a rigorous formal development method, using the specification language SPECTRUM. It demonstrates deductive software development starting from an axiomatic requirement specification and leading to an executable specification.

All proof obligations involved in the development process are indicated.

The second part in this volume is concerned with the *languages* used – and further developed – in the project. The first paper is this area, **KORSO Reference Languages – Concepts and Application Domains**, gives an overview of these languages and, at the same time, touches on nearly all the other topics, most particularly the case studies.

The next paper, **How to Cope with the Spectrum of SPECTRUM**, shows the syntactic and semantic connections of the specification language SPECTRUM with two other existing languages, which can be seen as executable subsets of SPECTRUM. This is followed by **A Fine-Grain Sort Discipline and Its Application to Formal Program Construction**, which describes in highly technical terms the synthesis of functional programs in the framework of a formally based development method. The aim here is to automate simple development steps by using automatic theorem-proving techniques.

The last paper in this part, **TROLL *light* – The Language and its Development Environment**, gives an informal outline of this specification language with its object-oriented features and special development and animation environment.

The papers that follow in the next part are more or less closely concerned with *development systems and logical frameworks.* The first paper in this part, **Formalization of Algebraic Specification in the Development Language DEVA** defines the language DEVA. It's purpose is to represent developments formally. With this one can formally describe the actions of a system. For the language SPECTRUM several refinement steps are formalized in the paper. There are many cross-connections between this paper and the following one, **Construction and Deduction Methods for the Formal Development of Software**, which gives the logical framework QED for formally describing specifications, programs and developments on the basis of a type theory.

The specification language OBSCURE and its environment are the subject of the next paper, **Experiences with a Specification Environment**. The focus here is on the prototypical executability of OBSCURE specifications.

In the area relating to *tools*, the emphasis is on deduction support. Different proof systems are presented, which were used and investigated in the course of the KORSO project. The first paper, **Towards Correct, Efficient and Reusable Transformational Developments** emphasizes the importance of development using transformation and so-called meta calculi.

The Verification System Tatzelwurm is concerned with the verification of sequential programs in a Pascal dialect, while in **SEDUCT – A Proof Com-**

piler for First Order Logic the characteristics and use of the so-called proof compiler are looked at in relation to concrete program development.

The verification system, TRAVERDI - **Transformation and Verification of Distributed Systems**, combines several interactive and automated proof techniques to a general tool for developing correct, distributed systems. It emphasizes the use of transformations for this task.

The last paper in this part, **The KIV Approach to Software Verification** describes the KIV (Karlsruhe Interactive Verifier) system, focusing specifically on the module system and the aspect of reuse. With this system several case studies were carried out.

The final part, *case studies*, reports on practical experience with the use of methods, languages and tools. The first paper of this part, **Three Selected Case Studies in Verification**, gives a description and motivation of the usage of the KIV system. By reference to three small examples for theorem proving a detailed productivity analysis for the KIV system is given.

Most of the research was carried out on the case study "Production Cell". Nearly all the partners were involved in this study, so that a separate report is devoted to it. The authors of the **Case Study "Production Cell": A Comparative Study in Formal Specification and Verification** therefore confine themselves to giving an overview of this work, while attempting to put the results into perspective.

Even the last paper in this volume, **The KORSO Case Study for Software Engineering with Formal Methods: A Medical Information System**, is a survey on work done in the biggest case study of the KORSO project. It focuses on requirements analysis supported by formal specification techniques. And we are proud of the main result of this case study: Formal methods scale up in a pleasant way.

To conclude, we would like to express our thanks to all those involved in the KORSO project, not only for their excellent work, but also for their exceptional cooperativeness and personal sense of identification with KORSO.

M. Broy, TU München,

S. Jähnichen TU Berlin

Table of Contents