Michael Butler
Michael G. Hinchey
María M. Larrondo-Petrie (Eds.)

LNCS 4789

Formal Methods and Software Engineering

9th International Conference
on Formal Engineering Methods, ICFEM 2007
Boca Raton, FL, USA, November 2007, Proceedings

Springer

TP311.5-53
F723.2
2007

Michael Butler   Michael G. Hinchey
María M. Larrondo-Petrie (Eds.)

Formal Methods and Software Engineering

9th International Conference
on Formal Engineering Methods, ICFEM 2007
Boca Raton, FL, USA, November 14-15, 2007
Proceedings

Springer

Volume Editors

Michael Butler
University of Southampton, School of Electronic and Computer Science
Highfield, Southampton, SO17 1BJ, UK
E-mail: m.j.butler@ecs.soton.ac.uk

Michael G. Hinchey
Loyola College in Maryland, Department of Computer Science
4501 N. Charles Street, Baltimore, MD 21210, USA
E-mail: mike.hinchey@usa.net

María M. Larrondo-Petrie
Florida Atlantic University, Department of Computer Science and Engineering
777 Glades Road SE-308, Boca Raton, FL 33431-0991, USA
E-mail: petrie@fau.edu

Library of Congress Control Number: 2007938401

CR Subject Classification (1998): D.2.4, D.2, D.3, F.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-76648-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-76648-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12187614 06/3180 5 4 3 2 1 0

Preface

Formal methods for the development of computer systems have been extensively researched and studied. A range of semantic theories, specification languages, design techniques, and verification methods and tools have been developed and applied to the construction of programs of moderate size that are used in critical applications. The challenge now is to scale up formal methods and integrate them into engineering development processes for the correct construction and maintenance of computer systems. This requires us to improve the state of the art by researching the integration of methods and their theories, and merging them into industrial engineering practice, including new and emerging practice.

ICFEM, the International Conference on Formal Engineering Methods, aims to bring together those interested in the application of formal engineering methods to computer systems. Researchers and practitioners, from industry, academia, and government, are encouraged to attend and to help advance the state of the art. The conference particularly encourages research that aims at a combination of conceptual and methodological aspects with their formal foundation and tool support, and work that has been incorporated into the production of real systems.

This volume contains the papers presented at ICFEM 2007 held November 14–15, 2007 in Florida Atlantic University, Boca Raton, Florida. There were 38 submissions. Each submission was reviewed by four Program Committee members. The committee decided to accept 19 papers based on originality, technical soundness, presentation, and relevance to formal engineering and verification methods. We thank the Program Committee members and the other referees for their effort and professional work in the reviewing and selecting process. The program also includes contributions from the two keynote speakers: Jean-Raymond Abrial and Tom Maibaum. Professor Abrial gave a talk on a system development process with Event-B and the Rodin Platform while Professor Maibaum gave a talk on the challenges of software certification.

A workshop on the *verifiable file store mini-challenge* was held on November 13, 2007 co-located with ICFEM 2007. This workshop was organized by Jim Woodcock and Leo Freitas as part of the Grand Challenge in Verified Software.

ICFEM 2007 was jointly organized and sponsored by Florida Atlantic University, Loyola College in Maryland, and the University of Southampton and we would like to thank all those who helped in the organization. We used the Easychair system to manage the submissions, refereeing, paper selection, and proceedings production. We would like to thank the Easychair team for a very powerful tool.

August 2007

Michael Butler
Mike Hinchey
Maria M. Larrondo-Petrie

Conference Organization

Conference Chair

General Chair	Mike Hinchey (Loyola College in Maryland, USA)
Program Chairs	Michael Butler (University of Southampton, UK)
	Maria M. Larrondo-Petrie (Florida Atlantic University, USA)
Publicity Chair	Denis Gracanin (Virginia Tech, USA)

Program Committee

Keijiro Araki
Farhad Arbab
David Basin
Ana Cavalcanti
Jessica Chen
Yoonsik Cheon
Kai Engelhardt
Eduardo B. Fernandez
Colin Fidge
John Fitzgerald
Marc Frappier
Marcelo Fabián Frias
Uwe Glässer
Joseph Kiniry
Shriram Krishnamurthi
Kung-Kiu Lau
Rustan Leino
Michael Leuschel
Xuandong Li
Zhiming Liu
Shaoying Liu
Tiziana Margaria
Huaikou Miao
Peter O'Hearn
Michael Poppleton
Marie-Laure Potet
Anders Ravn
Davide Sangiorgi
Mannu Satpathy
Klaus-Dieter Schewe
Kaisa Sere
Wuwei Shen
Marjan Sirjani
Ketil Stølen
Sofiene Tahar
Helen Treharne
T.H. Tse
Farn Wang
Wang Yi
Jian Zhang
Jin Song Dong
Zhenhua Duan

Local Organization

Eduardo B. Fernandez
Michael VanHilst
Nelly Delessy-Gassant
Maureen Manoly
Colleen Glazer

External Reviewers

Rezine Ahmed
Bernhard Aichernig
Joachim Baran
Achim D. Brucker
Neil Evans
Bernd Fischer
Wan Fokkink
Amjad Gawanmeh
Olga Grinchintein
Osman Hasan
Felix Klaedtke
István Knoll

Linas Laibinis	Sampath Prahlad	Volker Stolz
Yuan Fang Li	Zongyan Qiu	Jan Stöcker
Sotiris Moschoyiannis	S. Ramesh	Jun Sun
Juan Antonio Navarro-Pérez	Niloofar Razavi	Leonidas Tsiopoulos
Joseph Okika	Atle Refsdal	Andrzej Wasowski
Gennaro Parlato	Ragnhild Kobro Runde	Mohamed Zaki
Daniel Plagge	Mehrnoosh Sadrzadeh	Miaomiao Zhang
Marta Plaska	Mayank Saksena	Jianhua Zhao
	Corinna Spermann	Hui Liang

ICFEM Steering Committee

He Jifeng (East China Normal University, China)
Keijiro Araki (Kyushu University, Japan)
Jin Song Dong (National University, Singapore)
Chris George (UNU-IIST, Macao)
Mike Hinchey (Loyola College in Maryland, USA)
Shaoying Liu (Hosei University, Japan)
John McDermid (University of York, UK)
Tetsuo Tamai (University of Tokyo, Japan)
Jim Woodcock (University of York, UK)

¥560.00元

Table of Contents

A System Development Process with Event-B and the Rodin Platform

J.-R. Abrial

jabrial@inf.ethz.ch

Event-B is the name of a mathematical (set-theoretic) approach used to develop *complex discrete systems*, be they computerized or not.

The Rodin platform is an open tool set devoted to supporting the development of such systems. It contains a modeling database surrounded by various plug-ins: static checker, proof obligation generator, provers, model-checkers, animators, UML transformers, requirement document handler, etc. The database itself contains the various modeling elements needed to construct discrete transition system models: essentially variables, invariants, and transitions.

Formal Development. With the help of this palette, users can develop mathematical models and refine them. In doing so, they are able to reason, modify, and decompose their models before starting the effective implementation of the corresponding systems. Such an approach is well known and widely used in many mature engineering disciplines where reasoning on a abstract representation of the future system is routine. Just think of the usage of blueprints made by architects within a building construction process.

Technology Transfer. One of the main difficulties in transferring this technology is not that of its mastering by industry engineers (a common opinion shared by many analysts). It is rather, we think, the incorporation of this technology within the industrial *development process*. We believe that the above argument about the difficulty of mastering this technology is, in fact, a way of hiding (consciously or not) the one concerning the incorporation within the development process.

This Presentation. The aim of this presentation is to show that the Event-B technology can be put into practice. For this, we must follow a well defined development process. That process is precisely the one which has to be transferred to industry. The Rodin platform, in its final form, will be the supporting tool for achieving this.

Before describing the Event-B development process however, we make precise what we mean by an industrial development process in general terms.

Industrial Development Processes are now common practice among important critical system manufacturers (train signalling system companies, avionic and space companies, automotive manufacturers, power system designers, defense sector industries, etc.). A system development process contains the definition of the various milestones encountered in the system construction together with the precise definition of what is to be done between these milestones, by whom, and within which delays. It also contains different ways of re-iterating on these milestones in case the process encounters some difficulties.

Usually, industrial managers are very reluctant to modify their development processes because: (1) it is part of their company culture and image, (2) it is difficult to define and make precise, and (3) it is even more difficult to have them accepted and followed by working engineers.

In order to know how to modify the development process due to the introduction of some formal method technology (Event-B and Rodin) in the construction of complex systems, it is clearly very important to understand that this process is aimed at obtaining systems which can be considered to be *correct by construction*. This presentation does not pretend to solve all related problems nor to give the key to a successful incorporation of formal methods in industry: it aims at providing the beginning of a systematic way of envisaging these matters.

Let us now briefly present the Event-B development process and show how the Rodin platform supports it.

Requirement Document. After the initial feasibility studies phase which is not subsequently modified, the second phase of the process is the writing of the *requirement document*. It must be pointed out that most of the time such documents are very poor: quite often, they just contain the pseudo-solution of a problem which, to begin with, is not stated. Our opinion is that it is very risky to proceed further with such poor documents. More precisely, we think that it is necessary to rewrite them very carefully in a systematic fashion. Each requirement must be stated by means of a short English statement which is well recognizable and clearly labelled according to some taxonomy to be defined for each project.

The Rodin platform in its final form will provide a plug-in able to support the gradual construction of such structured requirement documents, to retrieve them, and to form the initial basis of the necessary traceability.

Refinement Strategy. The next phase consists in defining a temporary *refinement strategy*. It contains the successive steps of the refined models construction. Clearly, it is out of the question to construct a unique model taking account of all requirements at once. Each such refinement step must give a reference to the precise requirements, stated in the previous phase, which are taken into account. A preliminary completion study can be performed (no requirements are forgotten). The refinement strategy in this phase is only temporary as it might be reshaped in further phases.

The Rodin platform in its final form will provide a plug-in able to support the writing of the refinement strategy and to check that it is correctly linked to the requirement document.

Refinements and Proofs. The next phase is divided up in many sub-steps according to the precise strategy defined in the previous phase. Each sub-step is made of the definition of the *formal refinement* which is performed, together with the corresponding *proofs*. It might be accompanied by some model-checking, model testing, as well as animations activities.

The three previous activities are very important to be performed at each refinement sub-step as they help figuring out that some requirements are impossible to achieve (or very costly), whereas some other had been simply completely forgotten. In other words, these activities help *validate the requirement document.* The outcome of these activities (checked or tested properties and model animations) can be seen and understood by the "client", who is then able to judge whether what has been formally modeled at a given stage indeed corresponds to what he had in mind. Notice that in each refinement sub-step, it might be found also that the previous refinement strategy was not adequate so that it has to be modified accordingly.

The Rodin platform provides the core elements able to support this central phase: modeling database, proof obligation generator, and provers. The surrounding plug-ins (model-checker, animator, UML translator) support the other requirement document validation activities

Decomposition. The next phase proceeds with the *decomposition* of the refined model obtained at the end of the previous one. In particular, this decomposition might separate that part of the model dealing with the external environment from that part of the system dealing with the hardware or software implementation. The latter part might be refined in the same way as it was done on the global model in the previous phase. This refinement/decomposition pair might be repeated a number of times until one reaches a satisfactory architecture.

The Rodin platform in its final form will provide plug-ins to support and prove that proposed decompositions are correct.

Code Generation. The final phase consists in performing the various hardware or software *automatic code generation.*

The Rodin platform in its final form will provide plug-ins to perform these translations.

As can be seen the incorporation of these phases within an existing development process is certainly not an easy task. An important point to take into account is the incorporation (and measurement) of the many proofs which have to be performed in order to be sure that the final system will be indeed "correct by construction".

Challenges in Software Certification

Tom Maibaum

Software Quality Research Laboratory and Department of Computing and Software
McMaster University
1280 Main St West, Hamilton ON, Canada L8S 4K1
tom@maibaum.org

Abstract. As software has invaded more and more areas of everyday life, software certification has emerged as a very important issue for governments, industry and consumers. Existing certification regimes are generally focused on the wrong entity, the development process that produces the artifact to be certified. At best, such an approach can produce only circumstantial evidence for the suitability of the software. For proper scientific evaluation of an artifact, we need to address directly the attributes of the product and their acceptability for certification. However, the product itself is clearly not enough, as we need other artifacts, like requirements specifications, designs, test documentation, correctness proofs, etc. We can organise these artifacts using a simple, idealised process, in terms of which a manufacturer's own process can be "faked". The attributes of this idealised process and its products can be modelled, following the principles of Measurement Theory, using the product/process modelling method first introduced by Kaposi.

1 Introduction

Software standards have been a concern amongst the software engineering community for the past few decades and they remain a major focus today as a way of introducing and standardising engineering methods into the software industry. Software certification, or at least certification of systems including software, has emerged as an important issue for software engineers, industry, government and society. One has only to point to the many stories of serious disasters where software has been identified as the main culprit and the discomfort that is being felt about this amongst members of these communities. Several organisations, including standards organizations and licensing authorities, have published guidance documents to describe how software should be developed to meet standards or certification criteria.

In this paper, we focus on the issues related to software certification and refer to standards only when relevant, though much could be said about the failures of software related standards to meet criteria characterising rigorous engineering standards. These licensing organisations, through their guidance documents, aim to establish a common understanding between software producers and certifiers (evaluators). The US Food and Drug Administration (FDA) is one of these organisations. The Common Criteria consortium, focusing on security properties of IT systems, is another. The FDA has published several voluminous guidance documents concerning the validation of medical software (as has The Common Criteria consortium on security properties). However, these recommendations are not specified in an explicit and precise manner. In more detail, the FDA validation approach, as described in the FDA guidance document [6]:

- does not describe effectively the objects that are subject to assessment,
- does not specify the measurable attributes that characterize these objects, and
- does not describe the criteria on which the FDA staff will base their decision, in order to approve or reject the medical software and, therefore, does not describe the measurement procedures to be used to ascertain the values of the relevant attributes of the objects being assessed.

In fact, the focus of these documents is on the characteristics of a software development *process* that is likely to produce satisfactory software. It shares this approach and concern with almost all certification authorities' requirements (as well as those of standards organisations and approaches based on 'maturity', such as CMM [17]). This seems to miss the point of the aim of certification, namely to ascertain whether the *product*, for which a certificate is being sought, has appropriate characteristics. Certification should be a *measurement* based activity, in which an objective assessment of a product is made in terms of the values of measurable attributes of the product, using an agreed objective function. (This objective function, defined in terms of the measurable attributes of the product, is itself subjectively defined; but once agreed, its use is completely objective, predictable and, perhaps most importantly, repeatable.) After all, we are not going to be happy if an avoidable disaster is down to a product being faulty, even though the process that produced it was supposed to deliver a sound product. A process can never provide this guarantee, if it does not actually examine relevant qualities of the product being certified. Even if the process is one that gives us correctness by construction (in the sense used in formal methods), mere correctness is not enough to convince us of the acceptability of the product. (For example, the specification on which the correctness assertion is based may be faulty. Or not all requirements have been taken into account. See [15,22,23].)

Hence, our hypothesis, boldly stated, is that process oriented standards and certification regimes will never prove satisfactory as ways of guaranteeing software properties and providing a basis for licensing, and we have to develop a properly scientific, product based approach to certification.

2 Process Oriented Standards and Certification

The Food and Drug Administration (FDA) is a public agency in the United States of America concerned with the validation of medical device software or software used to design, develop, or produce medical devices in the United States. In response to the questions about FDA validation requirements, the FDA has expressed its current thinking about medical software validation through guidance documents [6.7.8]. These documents target both the medical software industry and FDA staff. According to the FDA, validation is an important activity that has to be undertaken throughout the software development lifecycle. In other words, it occurs at the beginning, end and even during stages of software development.

For example, the FDA guidance documents recommend validation to start early while the software is being developed. In this sense, the FDA guidance document [6] considers other activities; like planning, verification, testing, traceability, configuration management; as important activities which all together participate in reaching a conclusion that the software is validated.

In essence, the FDA validation approach is a generic approach. It appears in the form of recommendations to apply some software engineering practices. These practices are considered to be working hand by hand to support the validation process. The reason behind FDA taking such a generic approach is due to the 'variety of medical devices, processes, and manufacturing facilities' [6]. In other words, the nature of validation is significantly dependant on the medical device itself. Examples of such validation determinant factors are [6]: availability of production environment for validating the software, ability to simulate the production environment, availability of supportive devices, level of risk, any prerequisite regulations/approvals re validation, etc.

The recommendations in the FDA guidance documents aim to make it possible for the FDA to reach a conclusion that the software is validated. It applies to software [6]:

- used as a component, part, or accessory of a medical device;
- that is itself a medical device (e.g., blood establishment software);
- used in the production of a device (e.g., programmable logic controllers in manufacturing equipment);
- used in implementation of the device manufacturer's quality system (e.g., software that records and maintains the device history record).

Having reached the conclusion that the software is validated increases the level of confidence in the software and, accordingly, the medical device as well. In its guidance documents, the FDA recommends certain activities to be undertaken and certain deliverables to be prepared during the development of the medical software. These activities and deliverables are subject to validation. For instance, validating the Software Requirements Specification (SRS), a deliverable that contains all requirements, aims to ensure that there are no ambiguous, incomplete, unverifiable and technically infeasible requirements. Such validation seeks to ensure that these requirements essentially describe the user needs, and are sufficient to achieve the users' objectives. In the same manner, testing is another key activity that is thoroughly described in the guidance. On the other hand, the guidance points out some issues that are interrelated as a result of the nature of software. Examples of such issues are: frequent changes and their negative consequence, personnel turnover in the sense that software maintainers might have not be involved in the original development. Moreover, the FDA guidance stresses the importance of having well-defined procedures to handle any software change introduced. Validation in this context addresses the newly added software as well as already existing software. In other words, in addition to validating the newly added pieces (components) of code, the effect of these new pieces on the existing ones has to be checked. Such a check ensures that the new components have no negative impact on the existing ones. Furthermore, the guidance highlights the importance of having independence in the review process, in the sense that the personnel who participate in validating the software are not the ones who developed it.

These are mainly the kinds of issues which the FDA guidance documents address with regard to software validation. In terms of software development, the FDA