Relating Software Requirements and Architectures

Paris Avgeriou • John Grundy • Jon G. Hall •
Patricia Lago • Ivan Mistrík
Editors

Relating Software Requirements and Architectures

Springer

Editors
Paris Avgeriou
Department of Mathematics and Computing Science
University of Groningen
9747 AG Groningen
The Netherlands
paris@cs.rug.nl

Jon G. Hall
Open University
Milton Keynes MK7 6AA
United Kingdom
J.G.Hall@open.ac.uk

Ivan Mistrík
Werderstr. 45
69120 Heidelberg
Germany
i.j.mistrik@t-online.de

John Grundy
Swinburne University of Technology
Hawthorn, VIC 3122
Australia
jgrundy@swin.edu.au

Patricia Lago
Vrije Universiteit
Dept. Computer Science
De Boelelaan 1081 A
1081 HV Amsterdam
Netherlands
patricia@cs.vu.nl

ISBN 978-3-642-21000-6 e-ISBN 978-3-642-21001-3
DOI 10.1007/978-3-642-21001-3
Springer Heidelberg Dordrecht London New York
ACM Codes: D.2, K.6

Library of Congress Control Number: 2011935050

Foreword

Why have a book about the relation between requirements and architecture? Requirements provide the function for a system and the architecture provides the form and after all, to quote Louis Sullivan, "form follows function." It turns out not to be so simple. When Louis Sullivan was talking about function, he was referring to the flow of people in a building or the properties of the spaces in the various floors. These are what we in the software engineering field would call quality attributes, not functionality. Understanding the relation between requirements and architecture is important because the requirements whether explicit or implicit do represent the function and the architecture does determine the form. If the two are disconnected, then there is a fundamental problem with the system being constructed.

The figure below gives some indication of why the problem of relating requirements and architecture is a difficult one (Fig. 1).

- There are a collection of stakeholders all of whom have their own agendas and interests. Reconciling the stakeholders is a difficult problem.
- The set of requirements are shown as a cloud because some requirements are explicitly considered in a requirements specification document and other requirements, equally or more important, remain implicit. Determining all of the requirements pertaining to a system is difficult.
- The double headed arrow between the requirements and the architecture reflects that fact that requirements are constrained by what is possible or what is easy. Not all requirements can be realized within the time and budget allotted for a project.
- The feedback from the architecture to the stakeholders means that stakeholders will be affected by what they see from various versions of the system being constructed and this will result in changes to the requirements. Accommodating those changes complicates any development process.
- The environment in which the system is being developed will change. Technology changes, the legal environment changes, the social environment changes, and the competitive environment changes. The project team must decide the extent to which the system will be able to accommodate changes in the environment.

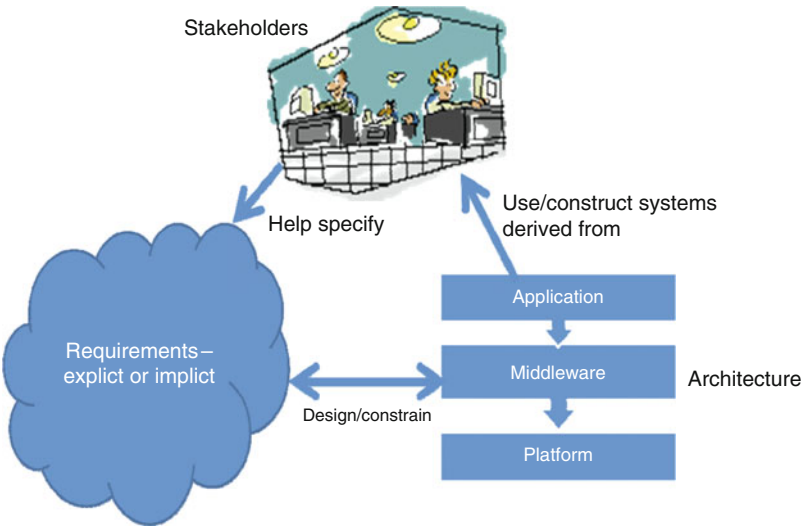

Fig. 1 A framing of the topics covered in this book

The authors in this book address the issues mentioned as well as some additional ones. But, as is characteristic of any rich problem area, they are not definitive. Some thoughts about areas that need further research are.

- Why not ask requirements engineers and architects what they see as the largest problems? This is not to say that the opinions expressed by the architects or requirements engineers would be more than an enumeration of symptoms for some deeper problem but the symptoms will provide a yardstick to measure research.
- What is the impact of scale? Architecture is most useful for large systems. To what extent are techniques for bridging the requirements/architecture gap dependent on scale? This leads to the question of how to validate any assertions. Assertions about software engineer ing methods or techniques are very difficult to validate because of the scale of the systems being constructed. One measure is whether a method or technique has been used by other than its author. This, at least, indicates that the method or technique is t ransferable and has some "face" validity.
- What aspects of the requirements/architecture gap are best spanned by tools and which by humans? Clearly tools are needed to manage the volume of requirements and to provide traceability and humans are needed to pe rform design work and elicit requirements from the stakeholders. But to what extent can tools support the human elements of the construction process that have to do with turning requirements into designs?
- Under what circumstances should the design/constrain arrow above point to the right and under what circumstances should it point to the left? Intuitively both directions have appeal but we ought to be able to make more precise statements

about when one does synthesis to generate a design and when one does analysis to determine the constraints imposed by the architectural decisions already made.

These are just some thoughts about the general problem. The chapters in this book provide much more detailed thoughts. The field is wide open, enjoy reading this volume and I encourage you to help contribute to bridging the requirements/architecture gap.

Len Bass
Software Engineering Institute
Pittsburgh, Pa, USA

Foreword

It must be the unique nature of software that necessitates the publication of a book such as this – in what other engineering discipline would it be necessary to argue for the intertwining of problem and solution, of requirements and architectures?

The descriptive nature of software may be one reason – software in its most basic form is a collection of descriptions, be they descriptions of computation or descriptions of the problems that the computation solves. After many years of focusing on software programs and their specifications, software engineering research began to untangle these descriptions, for example separating *what* a customer wants from *ho*w the software engineer delivers it. This resulted in at least two disciplines of description – requirements engineering and software architecture – each with their own representations, processes, and development tools. Recent years have seen researchers re-visit these two disciplines with a view to better understand the rich and complex relationships between them, and between the artifacts that they generate. Many of the contributions in this book address reflectively and practically these relationships, offering researchers and practicing software engineers tools to enrich and support software development in many ways: from the traditional way in which requirements – allegedly – precede design, to the pragmatic way in which existing solutions constrain what requirements can – cost-effectively – be met. And, of course, there is the messy world in between, where the customer needs change, where technology changes, and where knowledge about these evolves.

Amid these developments in software engineering, the interpretation of software has also widened – software is rarely regarded as simply a description that executes on a computer. Software permeates a wide variety of technical, socio-technical, and social systems, and while its role has never been more important, it no longer serves to solve the precise pre-defined problems that it once did. The problems that software solves often depend on what existing technology can offer, and this same technology can determine what problems can or should be solved. Indeed, existing technology may offer opportunities for solving problems that users never envisaged.

It is in this volatile and yet exciting context that this book seeks to make a particularly novel contribution. An understanding of the relationships between the problem world – populated by people and their needs – and the solution world – populated by systems and technology – is a pre-requisite for effective software engineering, and, more importantly, for delivering value to users.

The chapters in this book rightly focus on two sides of the equation – requirements and architectures – and the relationships between them. Requirements embody a range of problem world artifacts and considerations, from stakeholder goals to precise descriptions of the world that stakeholders seek to change. Similarly, architectures denote solutions – from the small executable program to the full structure and behavior of a software-intensive product line.

The ability to recognize, represent, analyze, and maintain the relationships between these worlds is, in many ways, the primary focus of this book. The editors have done an excellent job in assembling a range of contributions, rich in the breadth of their coverage of the research area, yet deep in addressing some of the fundamental research issues raised by 'real' world applications. And it is in this consideration of applications that their book differs from other research contributions in the area. The reason that relating requirements and architectures is an important research problem is that it is a problem that has its origins in the application world: requirements can rarely be expressed correctly or completely at the first attempt, and technical solutions play a large part in helping to articulate problems and to add value where value was hard to pre-determine. It is this 'messy' and changing real world that necessitates research that helps software engineers to deliver systems that satisfy, delight and even (pleasantly) surprise their customers. I am confident that, in this book, the editors have delivered a scholarly contribution that will evoke the same feelings of satisfaction, delight and surprise in its readers.

Lero (Ireland) & The Open University (UK) Bashar Nuseibeh
April 2011

Preface

This book brings together representative views of recent research and practice in the area of relating software requirements and software architectures. We believe that all practicing requirements engineers and software architects, all researchers advancing our understanding and support for the relationship between software requirements and software architectures, and all students wishing to gain a deeper appreciation of underpinning theories, issues and practices within this domain will benefit from this book.

Introduction

Requirements Engineering and Software Architecture have existed for at least as long as Software Engineering. However, only over the past 15–20 years have they become defined as sub-disciplines in their own right. Practitioners know that eliciting and documenting good requirements is still a very challenging problem and that it forms a heady mix with the difficulties inherent in architecting modern, complex heterogeneous software systems. Although research in each area is still active, it is in their combination that understanding is most actively sought. Presenting the current state of the art is the purpose of this book.

Briefly, for requirements engineering to have taken place, a set of requirements will have been elicited from often multiple stakeholders, analyzed for incompleteness, inconsistency and incorrectness, structured so they can be effectively shared with developers. As one would expect, large, complicated systems with a wide range of stakeholders – such as large banking, telecommunications and air traffic control systems, distributed development teams – have emerging and evolving requirements which make satisfaction very challenging. However, even smaller systems that are highly novel also present challenges to requirements engineers e.g., in the rapidly changing consumer electronics domain. As requirements are typically conceptualized and expressed initially in natural language by multiple stakeholders, identifying a set of consistent and complete requirements in a (semi-) formal model

for use by developers remains very challenging. These formalized requirements then must be satisfied in any architectural solution and the imprecision in moving from informal to formal requirements complicates this.

Good software architecture is fundamental to achieving successful software systems. Initially this meant defining the high level system decomposition combined with technological solutions to be used to meet requirements previously identified and codified. More recently this has grown to encompass the representation and use of architectural knowledge throughout the software engineering lifecycle, effective reuse of architectural styles and patterns, the engineering of self-adapting and self-healing autonomic system architectures, and the evolution of complex architectures over time. Without good architectural practices, software becomes as unstable and unreliable as construction without good foundational architecture: Quality attributes are not properly managed and risks are not mitigated.

What is Relating Software Requirements to Architecture?

Software requirements and architecture are intrinsically linked. Architecture solutions need to realize both functional and non-functional requirements captured and documented. While changes to a set of requirements may impact on its realizing architecture, choices made for an architectural solution may impact on requirements e.g. in terms of revising functional or non-functional requirements that can not actually be met.

A range of challenges present themselves when working on complex and emerging software systems in relating the system's requirements and its architecture. An incomplete list includes: When a requirement changes, what is the impact on its architecture? If we change decisions made during architecting, how might these impact on the system's requirements? How do we trace requirements to architecture elements and vice versa? How do new technologies impact on the feasibility of requirements? How do new development processes impact on the relationship between requirements and architecture e.g. agile, out sourcing, crowd sourcing etc? How do we relate an item in the requirements to its realizing features in the architecture and vice-versa, especially for large systems? How do we share requirements and architectural knowledge, especially in highly distributed teams? How do we inform architecture and requirements with economic as well as functional and non-functional software characteristics? How do practitioners currently solve these problems on large, real-world projects? What emerging research directions might assist in managing the relationship between software requirements and architecture?

In this book a range of approaches are described to tackle one or more of these issues. We interpret both "requirements" and "architecture" very widely. We also interpret the relationship between requirements and architecture very widely, encompassing explicit relationships between requirements and architecture partitions and constraints, to implicit relationships between decisions made in one

tangentially impacting on the other. Some of the contributions in this book describe processes, methods and techniques for representing, eliciting or discovering these relationships. Some describe technique and tool support for the management of these relationships through the software lifecycle. Other contributions describe case studies of managing and using such relationships in practice. Still others identify state of the art approaches and propose novel solutions to currently difficult or even intractable problems.

Book Overview

We have divided this book into four parts, with a general editorial chapter providing a more detailed review of the domain of software engineering and the place of relating software requirements and architecture. We received a large number of submissions in response to our call for papers and invitations for this edited book from many leading research groups and well-known practitioners of leading collaborative software engineering techniques. After a rigorous review process 15 submissions were accepted for this publication. We begin by a review of the history and concept of software engineering itself including a brief review of the discipline's genesis, key fundamental challenges, and we define the main issues in relating these two areas of software engineering.

Part I contains three chapters addressing the issue of requirements change management in architectural design through traceability and reasoning. Part II contains five chapters presenting approaches, tools and techniques for bridging the gap between software requirements and architecture. Part III contains four chapters presenting industrial case studies and artefact management in software engineering. Part IV contains three chapters addressing various issues such as synthesizing architecture from requirements, relationship between software architecture and system requirements, and the role of middleware in architecting for non-functional requirements. We finish with a conclusions chapter identifying key contributions and outstanding areas for future research and improvement of practice. In the sections below we briefly outline the contributions in each part of this book.

Part 1 – Theoretical Underpinnings and Reviews

The three chapters in this section identify a range of themes around requirements engineering. Collectively they build a theoretic framework that will assist readers to understand both requirements, architecture and the diverse range of relationships between them. They address key themes in the domain including change management, ontological reasoning and tracing between requirements elements and architecture elements, at multiple levels of detail.

Chapter 3 proposes Change-oriented Requirements Engineering (CoRE), a method to anticipate change by separating requirements into layers that change at relatively different rates. From the most stable to the most volatile, the authors identify several layers including: patterns, functional constraints, and business policies and rules. CoRE is empirically evaluated by the authors by applying it to a large-scale software system and then studying the observed requirements change from development to maintenance. Results show that CoRE accurately anticipates the relative volatility of the requirements and can thus help manage both requirements evolution but also derivative architectural change.

Chapter 4 introduces a general-purpose ontology that the authors have developed to address the problem of co-evolving requirements and architecture descriptions of a software system. They demonstrate an implementation of semantic wiki that supports traceability between elements in a co-evolving requirements specifications and corresponding architecture design. They demonstrate their approach using a reuse scenario and a requirements change scenario.

Chapter 5 investigates homogeneous and heterogeneous requirements traceability networks. These networks are achieved by using event-based traceability and call graphs. Both of these traces are harvested during a software project. These traceability networks can be used in understanding some of the resulting architectural styles observed based on the real time state of a software project. The authors demonstrate the utility of such traceability networks to monitor initial system decisions and identify bottlenecks in an exemplar software project, a pinball machine simulator.

Part 2 – Tools and Techniques

The five chapters in this section identify a range of themes around tools and techniques. Good tool support is essential to managing complex requirements and architectures on large projects. These tools must also be underpinned by techniques that provide demonstrative added value to developers. Techniques used range from goal-directed inference, uncertainty management, problem frames, service compositions, to quality attribute refinement from business goals.

Chapter 7 presents a goal-oriented software architecting approach, where functional requirements (FRs) and non-functional requirements (NFRs) are treated as goals to be achieved. These goals are then refined and used to explore achievement alternatives. The chosen alternatives and the goal model are then used to derive a concrete architecture by applying an architectural style and architectural patterns chosen based on the NFRs. The approach has been applied in an empirical study based on the well-known 1992 London ambulance dispatch system.

Chapter 8 describes a commitment uncertainty approach in which linguistic and domain-specific indicators are used to prompt for the documentation of perceived uncertainty. The authors provide structure and advice on the development process so that engineers have a clear concept of progress that can be made to reduce

technical risk. A key contribution is in the evaluation of the technique in the engine control domain. They show that the technique is able to suggest valid design approaches and that supported flexibility does accommodate subsequent changes to requirements. The authors' aim is not to replace the process of creating a suitable architecture but to provide a framework that emphasizes constructive design actions.

Chapter 9 presents a method to systematically derive software architectures from problem descriptions. The problem descriptions are set up using Jackson's problem frame approach. They include a context diagram describing the overall problem situation and a set of problem diagrams that describe sub-problems of the overall software development problem. The different sub-problems are the instances of problem frames and these are patterns for simple software development problems. Beginning from these pattern-based problem definitions, the authors derive a software architecture in three steps: an initial architecture contains one component for each sub-problem; they then apply different architectural and design patterns and introduce coordinator and facade components; and finally the components of the intermediate architecture are re-arranged to form a layered architecture and interface and driver components added. All artefacts are expressed using UML diagrams with specifically defined UML profiles. Their tool supports checking of different semantic integrity conditions concerning the coherence of different diagrams. The authors illustrate the method by deriving an architecture for an automated teller machine.

Chapter 10 proposes a solution to the problem of having a clear link between actual applications – also referred to as service compositions – and requirements the applications are supposed to meet. Their technique also stipulates that captured requirements must properly state how an application can evolve and adapt at runtime. The solution proposed in this chapter is to extend classical goal models to provide an innovative means to represent both conventional (functional and non-conventional) requirements along with dynamic adaptation policies. To increase support to dynamism, the proposal distinguishes between crisp goals, of which satisfiability is boolean, and fuzzy goals, which can be satisfied at different degrees. Adaptation goals are used to render adaptation policies. The information provided in the goal model is then used to automatically devise the application's architecture (i.e., composition) and its adaptation capabilities. The goal model becomes a live, runtime entity whose evolution helps govern the actual adaptation of the application. The key elements of this approach are demonstrated by using a service-based news provider as an exemplar application.

Chapter 11 presents a set of canonical business goals for organizations that can be used to elicit domain-specific business goals from various stakeholders. These business goals, once elicited, are used to derive quality attribute requirements for a software system. The results are expressed in a common syntax that presents the goal, the stakeholders for whom the goal applies, and the "pedigree" of the goal. The authors present a body of knowledge about business goals and then discuss several different possible engagement methods to use knowledge to elicit business goals and their relation to software architectural requirements. They describe a new

methodology to support this approach and describe their experiences applying it with an Air Traffic Management unit of a major U.S aerospace firm.

Part 3 –Industrial Case Studies

The four chapters in this section present several industrial case studies of relating software requirements and architecture. These range from developing next-generation Consumer Electronics devices with embedded software controllers, IT security software, a travel booking system, and various large corporate systems.

Chapter 13 addresses the problems in designing consumer electronics (CE) products where architectural description is required from an early stage in development. The creation of this description is hampered by the lack of consensus on high-level architectural concepts for the CE domain and the rate at which novel features are added to products. This means that old descriptions cannot simply be reused. This chapter describes both the development of a reference architecture that addresses these problems and the process by which the requirements and architecture are refined together. The reference architecture is independent of specific functionality and is designed to be readily adopted. The architecture is informed by information mined from previous developments and organized to be reusable in different contexts. The integrity between the roles of requirements engineer and architect, mediated through the reference architecture, is described and illustrated with an example of integrating a new feature into a mobile phone.

Chapter 14 presents a view-based, model-driven approach for ensuring the compliance ICT security issues in a business process of a large European company. Compliance in service-architectures means complying with laws and regulations applying to distributed software systems. The research question of this chapter is to investigate whether the authors' model-driven, view-based approach is appropriate in the context of this domain. This example domain can easily be generalized to many other problems of requirements that are hard to specify formally, such as compliance requirements, in other business domains. To this end, the authors present lessons learned as well metrics for measuring the achieved degree of separation of concerns and reduced complexity via their approach.

Chapter 15 proposes an approach to artifact management in software engineering that uses an artifact matrix to structure the artifact space of a project along with stakeholder viewpoints and realization levels. This matrix structure provides a basis on top of which relationships between artifacts, such as consistency constraints, traceability links and model transformations, can be defined. The management of all project artifacts and their relationships supports collaboration across different roles in the development process, change management and agile development approaches. The authors' approach is configurable to facilitate adaptation to different development methods and processes. It provides a basis to develop and/or to integrate generic tools that can flexibly support different methods. In particular, it can be leveraged to improve the transition from requirements analysis to

architecture design. The development of a travel booking system is used as an exemplar application domain.

Chapter 16 illustrates a set of proven practices as well as a conceptual methods that help software engineers classify and prioritize requirements which then serve as drivers for architecture design. The author claims that all design activities follow the approach of piecemeal growth. Rather than try and react against this, they urge in supporting this explicitly in the supporting requirements and architecting processes. Similarly, a layered approach has been found to be necessary and most effective when eliciting, documenting and managing these requirements and architectures. A method supporting this evolutionary growth of requirements and architecture is presented along with experiences of applying this approach on several SIEMENS projects.

Part 4 – Emerging Issues

The three chapters in this section address some emerging issues in the domain of relating software requirements and architecture. These issues include approaches to synthesizing candidate architectures from formal requirements descriptions, explicit, bi-directional constraint of requirements and architectural elements, and middleware-based, economic-informed definition of requirements.

Chapter 18 studies the generation of candidate software architectures from requirements using genetic algorithms. Architectural styles and patterns are used as mutations to an initial architecture and several architectural heuristics as fitness tests. The input for the genetic algorithm is a rudimentary architecture representing a very basic functional decomposition of the system, obtained as a refinement from use cases. This is augmented with specific modifiability requirements in the form of allowable change scenarios. Using a fitness function tuned for desired weights of simplicity, efficiency and modifiability, the technique produces a set of candidate architectural styles and patterns that satisfy the requirements. The quality of the produced architectures has been studied empirically by comparing generated architectures with ones produced by undergraduate students.

In Chap. 19 the authors claim that the relationship of a system's requirements and its architectural design is not a simple one. Previous thought has been that the requirements drive the architecture and the architecture is designed in order to meet requirements. In contrast, their experience is that a much more dynamic relationship needs to be achieved between these key activities within the system design lifecycle. This would then allow the architecture to constrain the requirements to an achievable set of possibilities, frame the requirements by making their implications on architecture and design clearer, and inspire new requirements from the capabilities of the system's architecture. The authors describe this rich interrelationship; illustrate it with a case study drawn from their experience; and present some lessons learned that they believe will be valuable for other software architects.

Chapter 20 discusses the problem of evolving non-functional requirements, their stability implications and economic ramifications on the software architectures induced by middleware. The authors look at the role of middleware in architecting for non-functional requirements and their evolution trends. They advocate adjusting requirements elicitation and management techniques to elicit not just the current non-functional requirements, but also to assess the way in which they will develop over the lifetime of the architecture and their economic ramifications. The range of these possible future requirements may then inform the selection of distributed components technologies, and subsequently the selection of application server products. They describe an economics-driven approach, based on the real options theory, which can assist in informing the selection of middleware to induce software architectures in relation to the evolving non-functional requirements. They review its application through a case study.

Current Challenges and Future Directions

We conclude this book with a chapter drawing conclusions from the preceding 15 chapters. We note many approaches adopt a goal-oriented paradigm to bridging the gap between requirements and architecture. Similarly, many techniques and tools attempt to address the problem of traceability between requirements elements and architectural decisions. The advance of reference architectures, patterns, and successful requirements models in many domains has assisted the development of complex requirements and architectures.

We observe, as have some of the authors of chapters in this book, that the waterfall software process heritage of "requirements followed by architecture" has probably always been a degree of fallacy, and is probably more so with today's complex and evolving heterogeneous software systems. Instead, viewing software requirements and software architecture as different viewpoints on the same problem may be a more useful future direction. Appropriate representation of architectural requirements and designs is still an area of emerging research and practice. This includes decisions, knowledge, abstractions, evolution and implications, not only technical but economic and social as well. To this end, some chapters and approaches touch on the alignment of business processes and economic drivers with technical requirements and architectural design decisions. While enterprise systems engineering has been an area of active practice and research for 30+ years, the relationship between business drivers and technical solutions is still ripe for further enhancement.

J. Grundy, I. Mistrík, J. Hall, P. Avgeriou, and P. Lago

Acknowledgements

The editors would like to sincerely thank the many authors who contributed their works to this collection. The international team of anonymous reviewers gave detailed feedback on early versions of chapters and helped us to improve both the presentation and accessibility of the work. Finally we would like to thank the Springer management and editorial teams for the opportunity to produce this unique collection of articles covering the wide range of issues in the domain of relating software requirements and architecture.

Contents

Contributors

Katrina Attwood Department of Computer Science, University of York, York YO10 5DD, United Kingdom, katrina.attwood@cs.york.ac.uk

Paris Avgeriou Department of Mathematics and Computing Science, University of Groningen, Groningen 9747 AG, The Netherlands, paris@cs.rug.nl

Rami Bahsoon University of Birmingham, School of Computer Science, Birmingham B15 2TT, United Kingdom, r.bahsoon@cs.bham.ac.uk

Luciano Baresi Dipartimento Elettronica e Informazione, Politecnico di Milano, Milan 20133, Italy, baresi@elet.polimi.it

Len Bass Software Engineering Institute/Carnegie Mellon University, Pittsburgh, PA 15213, USA, lenbass@cmu.edu

Kawtar Benghazi Departamento de Lenguajes y Sistemas Informaticos, Universidad de Granada, Granada 18071, Spain, benghazi@ugr.es

Stefan Biffl Vienna University of Technology, Institute of Software Technology and Interactive Systems, Wien 1040, Austria, stefan.biffl@tuwien.ac.at

Goetz Botterweck Lero – the Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland, goetz.botterweck@lero.ie

Christine Choppy Université Paris 13, LIPN, CNRS UMR 7030, Villetaneuse 93430, France, Christine.Choppy@lipn.univ-paris13.fr

Lawrence Chung Department of Computer Science, University of Texas at Dallas, Dallas, TX 75083, USA, chung@utdallas.edu

Paul Clements Software Engineering Institute/Carnegie Mellon University, Pittsburgh, PA 15213, USA, clements@sei.cmu.edu

Viktor Clerc VU University Amsterdam, FEW, Computer Science, Amsterdam 1081 HV, The Netherlands, viktor@cs.vu.nl

Schahram Dustar Vienna University of Technology, Information Systems Institute, Wien 1040, Austria, dustdar@infosys.tuwien.ac.at

Wolfgang Emmerich Department of Computer Science, University College London, London WC1E 6BT, United Kingdom, W.Emmerich@cs.ucl.ac.uk

Anthony Finkelstein Department of Computer Science, University College London, London WC1E 6BT, United Kingdom, a.finkelstein@cs.ucl.ac.uk

Jose Luis Garrido Departamento de Lenguajes y, Universidad de Granada Sistemas Informaticos, Granada 18071, Spain, jgarrido@ugr.es

John Grundy Swinburne University of Technology, Hawthorn, VIC 3122, Australia, jgrundy@swin.edu.au

Hadyatullah Kundi Department of Software Systems, Tampere University of Technology, Tampere, Finland, hadaytullah@tut.fi

Jon G Hall Open University, Milton Keynes MK7 6AA, United Kingdom, J.G.Hall@open.ac.uk

Denis Hatebur Universität Duisburg-Essen, Software Engineering, Duisburg 47057, Germany, denis.hatebur@uni-duisburg-essen.de

Maritta Heisel Universität Duisburg-Essen, Software Engineering, Duisburg 47057, Germany, maritta.heisel@uni-duisburg-essen.de

Ta'id Holmes Vienna University of Technology, Information Systems Institute, Wien 1040, Austria, tholmes@infosys.tuwien.ac.at

Maria V. Hurtado Departamento de Lenguajes y Sistemas Informaticos, Universidad de Granada, Granada 18071, Spain, mhurtado@ugr.es

Kai Koskimies Department of Software Systems, Tampere University of Technology, Tampere, Finland, kai.koskimies@tut.fi

Jochen M. Küster IBM Research – Zurich, Rüschlikon 8803, Switzerland, jku@zurich.ibm.com

Patricia Lago VU University Amsterdam, FEW, Computer Science, Amsterdam 1081 HV, The Netherlands, patricia@cs.vu.nl

Peng Liang Department of Mathematics and Computing Science, University of Groningen, Groningen 9747 AG, The Netherlands, liangp@cs.rug.nl

Soo Ling Lim Department of Computer Science, University College London, London WC1E 6BT, United Kingdom, s.lim@cs.ucl.ac.uk

Erkki Mäkinen Department of Computer Science, University of Tampere, Tampere, Finland, em@cs.uta.fi

John McDermid Department of Computer Science, University of York, York YO10 5DD, United Kingdom, john.mcdermid@cs.york.ac.uk

Ivan Mistrík Independent Consultant, Heidelberg 69120, Germany, i.j.mistrik@t-online.de

Manuel Noguera Departamento de Lenguajes y Sistemas Informaticos, Universidad de Granada, Granada 18071, Spain, mnoguera@ugr.es

Bashar Nuseibeh Lero – the Irish Software Engineering Research Centre and The Open University (UK), University of Limerick, Limerick, Ireland, Bashar. Nuseibeh@lero.ie

Inah Omoronya Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, inah.omoronyia@idi.ntnu.no

Liliana Pasquale Dipartimento Elettronica e Informazione, Politecnico di Milano, Milan 20133, Italy, pasquale@elet.polimi.it

Outi Räihä Department of Software Systems, Tampere University of Technology, Tampere, Finland, outi.raiha@tut.fi

Maria Luisa Rodriguez Departamento de Lenguajes y Sistemas Informaticos, Universidad de Granada, Granada 18071, Spain, mlra@ugr.es

Nick Rozanski Software Architect for a UK Investment Bank, London, United Kingdom, nick@rozanski.org.uk

Guttorm Sindre Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, guttorm.sindre@idi.ntnu.no

Michael Stal Siemens Corporate Technology, München, Germany and University of Groningen, Groningen, The Netherlands, michael.stal@gmail.com

Tor Stalhane Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, tor.stalhane@idi.ntnu.no

Zoe Stephenson Department of Computer Science, University of York, York Y010 5DD, United Kingdom, zoe@cs.york.ac.uk

Nary Subramanian Department of Computer Science, University of Texas at Tyler, Tyler, USA, Nary_Subramanian@UTTyler.edu

Sam Supakkul Department of Computer Science, University of Texas at Dallas, Dallas, USA, ssupakkul@ieee.org

Antony Tang Swinburne University of Technology, Hawthorn, VIC, 3122, Australia, ATang@groupwise.swin.edu.au

Huy Tran Vienna University of Technology, Information Systems Institute, Wien 1040, Austria, htran@infosys.tuwien.ac.at

Tim Trew Independent Embedded Software Architect and Software Specialist, Horley, Surrey, RH6 7BX, United Kingdom, tiptrew@theiet.org, tim.trew@btinternet.com

Hans van Vliet VU University Amsterdam, FEW, Computer Science, Amsterdam 1081 HV, The Netherlands, hans@cs.vu.nl

Hagen Völzer IBM Research – Zurich, Rüschlikon 8803, Switzerland, hvo@zurich.ibm.com

Eoin Woods Artechra, Hemel Hempstead, Herfordshire, United Kingdom, eoin.woods@artechra.com

Uwe Zdun Faculty of Computer Science, Vienna University of Technology, Wien 1090, Austria, zdun@infosys.tuwien.ac.at

Olaf Zimmermann IBM Research – Zurich, Rüschlikon 8803, Switzerland, olz@zurich.ibm.com