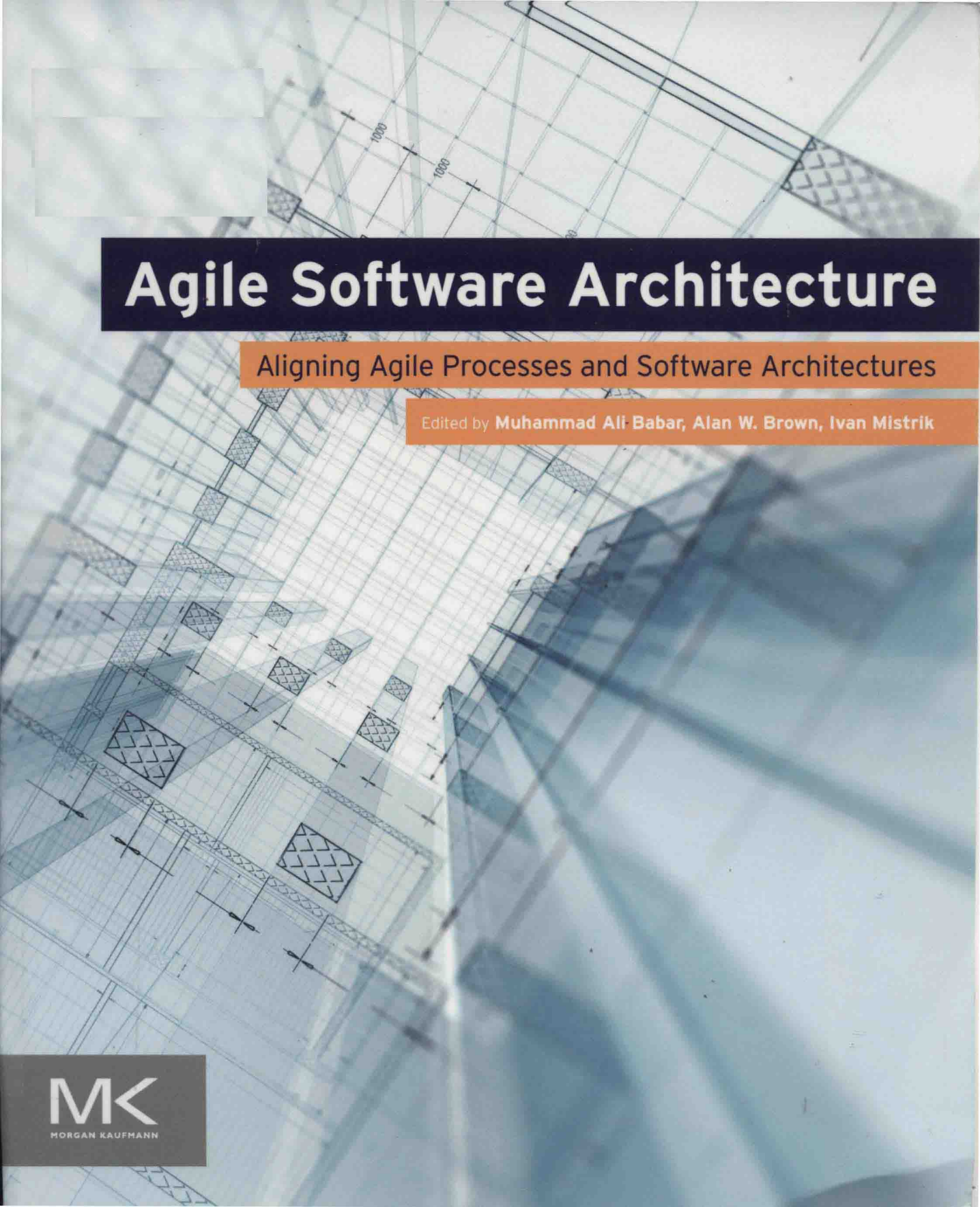

Agile Software Architecture

Aligning Agile Processes and Software Architectures

Edited by **Muhammad Ali Babar, Alan W. Brown, Ivan Mistrik**

MK
MORGAN KAUFMANN

Agile Software Architecture

Aligning Agile Processes and Software Architectures

Edited by

Muhammad Ali Babar

Alan W. Brown

Ivan Mistrik

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier

Acquiring Editor: *Todd Green*
Editorial Project Manager: *Lindsay Lawrence*
Project Manager: *Punithavathy Govindaradjane*
Designer: *Maria Inês Cruz*

Morgan Kaufmann is an imprint of Elsevier
225 Wyman Street, Waltham, MA 02451, USA

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods or professional practices, may become necessary. Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information or methods described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data

Agile software architecture : aligning agile processes and software architectures / edited by Muhammad Ali Babar, Alan W. Brown, Ivan Mistrik.

pages cm

Includes bibliographical references and index.

ISBN 978-0-12-407772-0 (pbk.)

1. Agile software development. 2. Software architecture. I. Ali Babar, Muhammad. II. Brown, Alan W., 1962- III. Mistrik, Ivan.

QA76.76.D47A3844 2013

005.1'2–dc23

2013040761

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

ISBN: 978-0-12-407772-0

This book has been manufactured using Print On Demand technology. Each copy is produced to order and is limited to black ink. The online version of this book will show color figures where appropriate.

For information on all MK publications visit our website at *www.mkp.com*

Agile Software Architecture

Acknowledgments

The editors would like to acknowledge the significant effort Kai Koskimies made during different phases of this book's editing phases. Judith Stafford also helped in framing the initial proposal for this book. We also sincerely thank many authors who contributed their works to this book. 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. Ali Babar worked on this project while based at Lancaster University UK and IT University of Copenhagen, Denmark. Finally, we would like to thank the Elsevier management and editorial teams, in particular to Todd Green and Lindsay Lawrence, for the opportunity to produce this unique collection of articles covering the wide range of areas related to aligning agile processes and software architectures.

About the Editors

MUHAMMED ALI BABAR

Dr. M. Ali Babar is a Professor of Software Engineering (Chair) at the School of Computer Science, the University of Adelaide, Australia. He also holds an Associate Professorship at IT University of Copenhagen, Denmark. Prior to this, he was a Reader in Software Engineering at Lancaster University UK. Previously, he worked as a researcher and project leader in different research centers in Ireland and Australia. He has authored/co-authored more than 140 peer-reviewed research papers in journals, conferences, and workshops. He has co-edited a book, *Software Architecture Knowledge Management: Theory and Practice*. Prof. Ali Babar has been a guest editor of several special issues/sections of *IEEE Software, JSS, ESEJ, SoSyM, IST,* and *REJ*. Apart from being on the program committees of several international conferences such as WICSA/ECSA, ESEM, SPLC, ICGSE, and ICSSP for several years, Prof. Ali Babar was the founding general chair of the Nordic-Baltic Symposium on Cloud Computing and Internet Technologies (NordiCloud) 2012. He has also been co-(chair) of the program committees of several conferences such as NordiCloud 2013, WICSA/ECSA 2012, ECSA2010, PROFES2010, and ICGSE2011. He is a member of steering committees of WICSA, ECSA, NordiCloud and ICGSE. He has presented tutorials in the areas of cloud computing, software architecture and empirical approaches at various international conferences. Prior to joining R&D field, he worked as a software engineer and an IT consultant for several years in Australia. He obtained a PhD in computer science and engineering from University of New South Wales, Australia.

ALAN W. BROWN

Alan W. Brown is Professor of Entrepreneurship and Innovation in the Surrey Business School, University of Surrey, UK. where he leads activities in the area of corporate entrepreneurship and open innovation models. In addition to teaching activities, he focuses on innovation in a number of practical research areas with regard to global enterprise software delivery, agile software supply chains, and the investigation of "open commercial" software delivery models. He has formerly held a wide range of roles in industry, including Distinguished Engineer and CTO at IBM Rational, VP of Research at Sterling Software, Research Manager at Texas Instruments Software, and Head of Business Development in a Silicon Valley startup. In these roles Alan has worked with teams around the world on software engineering strategy, process improvement, and the transition to agile delivery approaches. He has published over 50 papers and written four books. He holds a Ph.D. in Computing Science from the University of Newcastle upon Tyne, UK.

IVAN MISTRIK

Ivan Mistrik is a computer scientist who is interested in system and software engineering (SE/SWE) and in system and software architecture (SA/SWA); in particular, he is interested in life cycle system/software engineering, requirements engineering, relating software requirements and architectures, knowledge management in software development, rationale-based software development, aligning enterprise/system/software architectures, and collaborative system/software engineering. He has more than forty years' experience in the field of computer systems engineering as an information systems developer, R&D leader, SE/SA research analyst, educator in computer sciences, and ICT management consultant. In the past 40 years, he has worked primarily at various R&D institutions and has consulted on a variety of large international projects sponsored by ESA, EU, NASA, NATO, and UN. He has also taught university-level computer sciences courses in software engineering, software architecture, distributed information systems, and human-computer interaction. He is the author or co-author of more than 80 articles and papers that have been published in international journals and books and presented at international conferences and workshops; most recently, he wrote the chapter "Capture of Software Requirements and Rationale through Collaborative Software Development" in the book *Requirements Engineering for Sociotechnical Systems*, the paper "Knowledge Management in the Global Software Engineering Environment," and the paper "Architectural Knowledge Management in Global Software Development." He has also written over 90 technical reports and presented over 70 scientific/technical talks. He has served on many program committees and panels of reputable international conferences and organized a number of scientific workshops, most recently two workshops on Knowledge Engineering in Global Software Development at the International Conference on Global Software Engineering 2009 and 2010. He has been a guest editor of *IEE Proceedings Software: A Special Issue on Relating Software Requirements and Architectures,* published by IEE in 2005. He has also been lead editor of the book *Rationale Management in Software Engineering*, published in 2006; the book *Collaborative Software Engineering*, published in 2010; and the book *Relating Software Requirements and Architectures,* published in 2011. He has also co-authored the book *Rationale-Based Software Engineering*, published in May 2008. He is a lead editor of the *Expert Systems Special Issue on Knowledge Engineering in Global Software Development* to be published in 2012, and he has organized the IEEE International Workshop on the Future of Software Engineering for/in the Cloud (FoSEC) that was held in conjunction with IEEE Cloud 2011. He was a guest editor of the *Journal of Systems and Software Special Issue on the Future of Software Engineering for/in the Cloud* in 2013 and a lead editor of the book on *Aligning Enterprise, System, and Software Architectures* to be published in 2012.

List of Contributors

Sarah Al-Azzani
University of Birmingham, Birmingham, UK

Ahmad Al-Natour
University of Birmingham, Birmingham, UK

Paris Avgeriou
University of Groningen, Groningen, The Netherlands

Muhammad Ali Babar
The University of Adelaide, Adelaide, SA, Australia

Rami Bahsoon
University of Birmingham, Birmingham, UK

Kawtar Benghazi
Universidad de Granada, Granada, Spain

Jan Bosch
Chalmers University of Technology, Gothenburg, Sweden

Georg Buchgeher
Software Competence Center Hagenberg (SCCH), Hagenberg, Austria

Lawrence Chung
University of Texas at Dallas, Richardson, TX, USA

James O. Coplien
Gertrud & Cope, Espergærde, Denmark

Jane Cleland-Huang
DePaul University, Chicago, IL, USA

Adam Czauderna
DePaul University, Chicago, IL, USA

Jessica Díaz
Universidad Politécnica de Madrid (Technical U. of Madrid), Madrid, Spain

Peter Eeles
IBM, London, UK

Veli-Pekka Eloranta
Tampere University of Technology, Tampere, Finland

Uwe Friedrichsen
Codecentric AG, Solingen, Germany

Matthias Galster
University of Canterbury, Christchurch, New Zealand

Juan Garbajosa
Universidad Politécnica de Madrid (Technical U. of Madrid), Madrid, Spain

Stephen Harcombe
Northwich, Cheshire, UK

Richard Hopkins
IBM, Cleveland, UK

Ben Isotta-Riches
Aviva, Norwich, UK

Kai Koskimies
Tampere University of Technology, Tampere, Finland

José Luis Garrido
Universidad de Granada, Granada, Spain

Mehdi Mirakhorli
DePaul University, Chicago, IL, USA

Manuel Noguera
Universidad de Granada, Granada, Spain

Jennifer Pérez
Universidad Politécnica de Madrid (Technical U. of Madrid), Madrid, Spain

Janet Randell
Aviva, Norwich, UK

Trygve Reenskaug
University of Oslo, Oslo, Norway

Antonio Rico
Universidad de Granada, Granada, Spain

Jan Salvador van der Ven
Factlink, Groningen, The Netherlands

Michael Stal
Siemens AG, Corporate Research & Technology, Munich, Germany

Rainer Weinreich
Johannes Kepler University Linz, Linz, Austria

Agustín Yagüe
Universidad Politécnica de Madrid (Technical U. of Madrid), Madrid, Spain

Foreword by John Grundy

Architecture vs Agile: competition or cooperation?

Until recently, conventional wisdom has held that software architecture design and agile development methods are somehow "incompatible," or at least they generally work at cross-purposes [1]. Software architecture design has usually been seen by many in the agile community as a prime example of the major agile anti-pattern of "big design up front." On the other hand, agile methods have been seen by many of those focusing on the discipline of software architecture as lacking sufficient forethought, rigor, and far too dependent on "emergent" architectures (a suitable one of which may never actually emerge). In my view, there is both a degree of truth and a substantial amount of falsehood in these somewhat extreme viewpoints. Hence, the time seems ripe for a book exploring leading research and practice in an emerging field of "agile software architecture," and charting a path for incorporating the best of both worlds in our engineering of complex software systems.

In this foreword, I briefly sketch the background of each approach and the anti-agile, anti-software architecture viewpoints of both camps, as they seem to have become known. I deliberately do this in a provocative and all-or-nothing way, mainly to set the scene for the variety of very sensible, balanced approaches contained in this book. I hope to seed in the reader's mind both the traditional motivation of each approach and how these viewpoints of two either-or, mutually exclusive approaches to complex software systems engineering came about. I do hope that it is apparent that I myself believe in the real benefits of both approaches and that they are certainly in no way incompatible; agile software architecting—or architecting for agile, if you prefer that viewpoint—is both a viable concept and arguably the way to approach the current practice of software engineering.

SOFTWARE ARCHITECTURE—THE "TRADITIONAL" VIEW

The concept of "software architecture"—both from a theoretical viewpoint as a means of capturing key software system structural characteristics [2] and practical techniques to develop and describe [3, 4]—emerged in the early to mid-1980s in response to the growing complexity and diversity of software systems. Practitioners and researchers knew implicitly that the concept of a "software architecture" existed in all but the most trivial systems. Software architecture incorporated elements including, but not limited to, human machine interfaces, databases, servers, networks, machines, a variety of element interconnections, many diverse element properties, and a variety of further structural and behavioral subdivisions (thread

management, proxies, synchronization, concurrency, real-time support, replication, redundancy, security enforcement, etc.). Describing and reasoning about these elements of a system became increasingly important in order to engineer effective solutions, with special purpose "architecture description languages" and a wide variety of architecture modeling profiles for the Unified Modeling Language (UML). Software architecting includes defining an architecture from various perspectives and levels of abstraction, reasoning about the architecture's various properties, ensuring the architecture is realizable by a suitable implementation which will meet system requirements, and evolving and integrating complex architectures.

A number of reusable "architecture patterns" [3] have emerged, some addressing quite detailed concerns (e.g., concurrency management in complex systems), with others addressing much larger-scale organizational concerns (e.g., multitier architectures). This allowed a body of knowledge around software architecture to emerge, allowing practitioners to leverage best-practice solutions for common problems and researchers to study both the qualities of systems in use and to look for improvements in software architectures and architecture engineering processes.

The position of "software architecting" in the software development lifecycle was (and still is) somewhat more challenging to define. Architecture describes the solution space of a system and therefore traditionally is thought of as an early part of the design phase [3, 4]. Much work has gone into developing processes to support architecting complex systems, modeling architectures, and refining and linking architectural elements into detailed designs and implementations. Typically, one would identify and capture requirements, both functional and nonfunctional, and then attempt to define a software architecture that meets these requirements.

However, as all practitioners know, this is far easier said than done for many real-world systems. Different architectural solutions themselves come with many constraints—which requirements can be met and how they can be met, particularly nonfunctional requirements, are important questions. Over-constrained requirements may easily describe a system that has no suitable architectural realization. Many software applications are in fact "systems of systems" with substantive parts of the application already existent and incorporating complex, existent software architecture that must be incorporated. In addition, architectural decisions heavily influence requirements, and coevolution of requirements and architecture is becoming a common approach [5]. Hence, software architectural development as a top-down process is under considerable question.

AGILE METHODS—THE "TRADITIONAL" VIEW

The focus in the 1980s and 90s on extensive up-front design of complex systems, development of complex modeling tools and processes, and focus on large investment in architectural definition (among other software artifacts) were seen by many to have some severe disadvantages [6]. Some of the major ones identified included

over-investment in design and wasted investment in over-engineering solutions, inability to incorporate poorly defined and/or rapidly changing requirements, inability to change architectures and implementations if they proved unsuitable, and lack of a human focus (both customer and practitioner) in development processes and methods. In response, a variety of "agile methods" were developed and became highly popular in the early to mid- 2000s. One of my favorites and one that I think exemplifies the type is Kent Beck's eXtreme Programming (XP) [7].

XP is one of many agile methods that attempt to address these problems all the way from underlying philosophy to pragmatic deployed techniques. Teams comprise both customers and software practitioners. Generalist roles are favored over specialization. Frequent iterations deliver usable software to customers, ensuring rapid feedback and continuous value delivery. Requirements are sourced from focused user stories, and a backlog and planning game prioritizes requirements, tolerating rapid evolution and maximizing value of development effort. Test-driven development ensures requirements are made tangible and precise via executable tests. In each iteration, enough work is done to pass these tests but no more, avoiding over-engineering. Supporting practices, including 40-hour weeks, pair programming, and customer-on-site avoid developer burnout, support risk mitigation and shared ownership, and facilitate human-centric knowledge transfer.

A number of agile approaches to the development of a "software architecture" exist, though most treat architecture as an "emergent" characteristic of systems. Rather than the harshly criticized "big design up front" architecting approaches of other methodologies, spikes and refactoring are used to test potential solutions and continuously refine architectural elements in a more bottom-up way. Architectural spikes in particular give a mechanism for identifying architectural deficiencies and experimenting with practical solutions. Refactoring, whether small-scale or larger-scale, is incorporated into iterations to counter "bad smells,"—which include architectural-related problems including performance, reliability, maintainability, portability, and understandability. These are almost always tackled on a need-to basis, rather than explicitly as an up-front, forward-looking investment (though they of course may bring such advantages).

SOFTWARE ARCHITECTURE—STRENGTHS AND WEAKNESSES WITH REGARD TO AGILITY

Up-front software architecting of complex systems has a number of key advantages [8]. Very complex systems typically have very complex architectures, many components of which may be "fixed" as they come from third party systems incorporated into the new whole. Understanding and validating a challenging set of requirements may necessitate modeling and reasoning with a variety of architectural solutions, many of which may be infeasible due to highly constrained requirements. Some requirements may need to be traded off against others to even make the overall

system feasible. It has been found in many situations to be much better to do this in advance of a large code base and complex architectural solution to try and refactor [8]. It is much easier to scope resourcing and costing of systems when a software architecture that documents key components exists upfront. This includes costing nonsoftware components (networks, hardware), as well as necessary third party software licenses, configuration, and maintenance.

A major criticism of upfront architecting is the potential for over-engineering and thus over-investment in capacity that may never be used. In fact, a similar criticism could be leveled in that it all too often results in an under-scoped architecture and thus under-investing in required infrastructure, one of the major drivers in the move to elastic and pay-as-you-go cloud computing [9]. Another major criticism is the inability to adapt to potentially large requirements changes as customers reprioritize their requirements as they gain experience with parts of the delivered system [6]. Upfront design implies at least some broad requirements—functional and nonfunctional—that are consistent across the project lifespan. The relationship between requirements and software architecture has indeed become one of mutual influence and evolution [5].

AGILE—STRENGTHS AND WEAKNESSES WITH REGARD TO SOFTWARE ARCHITECTURE

A big plus of agile methods is their inherent tolerance—and, in fact, encouragement—of highly iterative, changeable requirements, focusing on delivering working, valuable software for customers. Almost all impediments to requirements change are removed; in fact, many agile project-planning methods explicitly encourage reconsideration of requirements and priorities at each iteration review—the mostly widely known and practiced being SCRUM [10]. Architectural characteristics of the system can be explored using spikes and parts found wanting refactored appropriately. Minimizing architectural changes by focusing on test-driven development—incorporating appropriate tests for performance, scaling, and reliability—goes a long way to avoiding redundant, poorly fitting, and costly over-engineered solutions.

While every system has a software architecture, whether designed-in or emergent, experience has shown that achieving a suitably complex software architecture for large-scale systems is challenging with agile methods. The divide-and-conquer approach used by most agile methods works reasonably well for small and some medium-sized systems with simple architectures. It is much more problematic for large-scale system architectures and for systems incorporating existent (and possibly evolving!) software architectures [8]. Test-driven development can be very challenging when software really needs to exist in order to be able to define and formulate appropriate tests for nonfunctional requirements. Spikes and refactoring support small-system agile architecting but struggle to scale to large-scale or even medium-scale architecture evolution. Some projects even find iteration sequences

become one whole refactoring exercise after another, in order to try and massively reengineer a system whose emergent architecture has become untenable.

BRINGING THE TWO TOGETHER—AGILE ARCHITECTING OR ARCHITECTING FOR AGILE?

Is there a middle ground? Can agile techniques sensibly incorporate appropriate levels of software architecture exploration, definition, and reasoning, before extensive code bases using an inappropriate architecture are developed? Can software architecture definition become more "agile," deferring some or even most work until requirements are clarified as develop unfolds? Do some systems best benefit from some form of big design up front architecting but can then adopt more agile approaches using this architecture? On the face of it, some of these seem counter-intuitive and certainly go against the concepts of most agile methods and software architecture design methods.

However, I think there is much to be gained by leveraging strengths from each approach to mitigate the discovered weaknesses in the other. Incorporating software architecture modeling, analysis, and validation in "architectural spikes" does not seem at all unreasonable. This may include fleshing out user stories that help to surface a variety of nonfunctional requirements. It may include developing a variety of tests to validate that these requirements are met. If a system incorporates substantive existing system architecture, exploring interaction with interfaces and whether the composite system meets requirements by appropriate test-driven development seems like eminently sensible early-phase, high-priority work. Incorporating software architecture-related stories as priority measures in planning games and SCRUM-based project management also seems compatible with both underlying conceptual models and practical techniques. Emerging toolsets for architecture engineering, particularly focusing on analyzing nonfunctional properties, would seem to well support and fit agile practices.

Incorporating agile principles into software architecting processes and techniques also does not seem an impossible task, whether or not the rest of a project uses agile methods. Iterative refinement of an architecture—including some form of user stories surfacing architectural requirements, defining tests based on these requirements, rapid prototyping to exercise these tests, and pair-based architecture modeling and analysis—could all draw from the demonstrated advantages of agile approaches. A similar discussion emerges when trying to identify how to leverage design patterns and agile methods, user-centered design and agile methods, and model-driven engineering and agile methods [1, 11, 12]. In each area, a number of research and practice projects are exploring how the benefits of agile methods might be brought to these more "traditional" approaches to software engineering, and how agile approaches might incorporate well-known benefits of patterns, User Centered Design (UCD), and Model Driven Engineering (MDE).

LOOKING AHEAD

Incorporating at least some rigorous software architecting techniques and tools into agile approaches appears—to me, at least—to be necessary for successfully engineering many nontrivial systems. Systems made up of architectures from diverse solutions with very stringent requirements, particularly challenging, nonfunctional ones, really need careful look-before-you-leap solutions. This is particularly so when parts of the new system or components under development may adversely impact existing systems (e.g., introduce security holes, privacy breaches, or adversely impact performance, reliability, or robustness). Applying a variety of agile techniques—and the philosophy of agile—to software architecting also seems highly worthwhile. Ultimately, the purpose of software development is to deliver high-quality, on-time, and on-budget software to customers, allowing for some sensible future enhancements. A blend of agile focus on delivery, human-centric support for customers and developers, incorporating dynamic requirements, and—where possible—avoiding over-documenting and over-engineering exercises, all seem to be of benefit to software architecture practice.

This book goes a long way toward realizing these trends of agile architecting and architecting for agile. Chapters include a focus on refactoring architectures, tailoring SCRUM to support more agile architecture practices, supporting an approach of continuous architecture analysis, and conducting architecture design within an agile process. Complementary chapters include analysis of the emergent architecture concept, driving agile practices by using architecture requirements and practices, and mitigating architecture problems found in many conventional agile practices.

Three interesting works address other topical areas of software engineering: engineering highly adaptive systems, cloud applications, and security engineering. Each of these areas has received increasing attention from the research and practice communities. In my view, all could benefit from the balanced application of software architecture engineering and agile practices described in these chapters.

I do hope that you enjoy this book as much as I enjoyed reading over the contributions. Happy agile software architecting!

John Grundy
Swinburne University of Technology,
Hawthorn, Victoria, Australia

References

[1] Nord RL, Tomayko JE. Software architecture-centric methods and agile development. IEEE Software 2006;23(2):47–53.

[2] Garlan D, Shaw M. Software architecture: perspectives on an emerging discipline. Angus & Robertson; 1996.

[3] Bass L, Clements P, Kazman R. Software architecture in practice. Angus & Robertson; 2003.