

Undergraduate Topics in Computer Science

Series Editor

Ian Mackie

Advisory Board

Samson Abramsky, University of Oxford, Oxford, UK

Karin Breitman, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil

Chris Hankin, Imperial College London, London, UK

Dexter C. Kozen, Cornell University, Ithaca, USA

Andrew Pitts, University of Cambridge, Cambridge, UK

Hanne Riis Nielson, Technical University of Denmark, Kongens Lyngby, Denmark

Steven S. Skiena, Stony Brook University, Stony Brook, USA

Iain Stewart, University of Durham, Durham, UK

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at <http://www.springer.com/series/7592>

Neil Walkinshaw

Software Quality Assurance

Consistency in the Face of Complexity and
Change

Neil Walkinshaw
Department of Computer Science
University of Leicester
Leicester
UK

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-64821-7 ISBN 978-3-319-64822-4 (eBook)
DOI 10.1007/978-3-319-64822-4

Library of Congress Control Number: 2017947829

© Springer International Publishing AG 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To Emma, Iona, and Dougie.

Preface

“Let’s think the unthinkable, let’s do the undoable. Let us prepare to grapple with the inef-fable itself, and see if we may not eff it after all.”

Douglas Adams, *Dirk Gently’s Holistic Detective Agency*

This book is an introduction for students to the main principles and some of the most popular techniques that constitute ‘software quality assurance’. It is worth emphasising from the outset that this book is *not* a reference book. There are already plenty of excellent comprehensive Software Engineering reference books in print.

Instead, this book seeks to provide a focus on Quality Assurance that typical, more generic Software Engineering reference books do not. The goal is to do so in such a way that the book can be read from cover to cover throughout the course of a typical university module. Specifically, this book aims to be:

- **Concise:** It aims to be small enough to be readable in its entirety over the course of a typical software engineering module.
- **Explanatory:** When topics are covered, it is important not merely to describe *what* they are, but also *why* they are the way they are – describing what events, technologies, and individuals or organisations helped to shape them into what they are now.
- **Applied:** Topics will be covered with a view to giving the reader a good idea of how they can be applied in practice, and by pointing where possible to evidence about their efficacy.

Quality Assurance is often presented and discussed in somewhat utilitarian terms, as a set of necessary, occasionally tedious, techniques; required reading for anybody who aspires to become a capable, reliable Software Engineer. This brings us to the final, slightly more nebulous objective of this book: To convince the reader that there is much, much more to Quality Assurance than that.

We inhabit a world in which software is increasingly pervasive – controlling everything from light bulbs in homes to smart phones, cars, planes, power stations, and

voting machines. Failures in software quality can have and have had disastrous consequences. There is an urgent need for a widespread appreciation of how precarious software quality can be, and how it can improved and ensured.

Although the application of Quality Assurance techniques can become ‘tedious’, this misses what are (for the author at least) the real attractions. The subject is not only necessary, but academically fascinating too. There is no way of *guaranteeing* that a software system will ‘succeed’ - that it will not contain bugs, satisfy the customer, and be delivered on time and at cost. The task of building complex systems according to complex, continuously changing requirements, in a limited amount of time, within a limited budget, whilst managing large teams of developers, is enormously challenging. There is no single ‘best’ solution, and there are so many open (often surprising) problems.

Acknowledgements

This book is an extension of the course notes for the “Software Quality Assurance and Metrics” course at the University of Leicester, jointly taught to under- and postgraduate students. The course was originally taught by Helge Janicke (now at De Montfort University) until I took over as convenor in 2013. Although the course has changed in several respects, I am very grateful to Helge for developing the initial structure, and thus setting the direction for a large portion of the subject-matter covered in this book.

Throughout the writing of the book, several undergraduate and postgraduate students have been kind enough to provide valuable feedback on its contents. Many thanks especially to Cara Bateman, Anita Lad, Shriya Malhotra, and Sylvester Saracevas.

Finally, I owe a debt of gratitude to the editorial team at Springer, and Ralf Gerstner in particular, for their valuable support, feedback, and patience.

Contents

- 1 Introduction** 1
 - 1.1 Consistency, Complexity, and Change 1
 - 1.2 Synopsis 2
- 2 What Is Software Quality, and Why Does it Matter?** 7
 - 2.1 Why Care about Software Quality? 7
 - 2.2 What Drives Software Quality Assurance? 14
 - 2.3 Defining “Software Quality” 16
 - 2.3.1 The Challenge of Defining Quality 16
 - 2.3.2 Quality Models - a Historical Perspective 18
 - 2.4 Key Points 21
- 3 Software Development Processes and Process Improvement** 23
 - 3.1 Process and Process Improvement in Manufacturing 24
 - 3.1.1 The Industrial Revolution 24
 - 3.1.2 Plan Do Check Act 26
 - 3.1.3 Quality-Driven Manufacturing in Japan 27
 - 3.1.4 Total Quality Management 30
 - 3.2 The Software Development Process 31
 - 3.2.1 The Waterfall Model 33
 - 3.2.2 Iterative and Incremental Software Development 35
 - 3.3 Agile Software Development 38
 - 3.3.1 The Principles of Agile Software Development 38
 - 3.3.2 An Example: SCRUM 39
 - 3.3.3 Relation to Total Quality Management 42
 - 3.3.4 Why Not Always Go Agile? 44
 - 3.4 Software Process Improvement - The Capability Maturity Model ... 45
 - 3.5 Key Points 48

4	Managing Requirements and Code	51
4.1	Managing Requirements	51
4.1.1	What is a Requirement?	52
4.1.2	Requirements Elicitation	53
4.1.3	Requirements Documents	56
4.1.4	Security Requirements	59
4.1.5	Tracing Requirements	60
4.1.6	Prioritisation	62
4.1.7	Oversight with Kanban boards	64
4.2	Writing Maintainable Source Code and Handling Change	64
4.2.1	Coding Conventions and Design / Architecture Patterns	65
4.2.2	Collaborative Development and Version Repositories	69
4.3	Key Points	74
5	Planning Activities and Predicting Costs	77
5.1	Planning	78
5.1.1	Program Evaluation and Review Technique (PERT)	78
5.1.2	Gantt Charts	81
5.2	Predicting Costs	82
5.2.1	Base Models	82
5.2.2	Parameter Fitting by Linear Regression	83
5.2.3	COCOMO	84
5.2.4	Planning Poker	90
5.2.5	Uncertainty and Predictive Accuracy	91
5.2.6	Keeping Track of Progress	92
5.3	Key Points	94
6	Testing	95
6.1	The Foundations of Software Testing	95
6.2	White-Box Testing	99
6.2.1	Code coverage	99
6.2.2	White Box Test Generation	101
6.2.3	The Case(s) Against Code Coverage	106
6.2.4	Goto Fail: A Case For Code Coverage	108
6.2.5	An Alternative: Mutation Testing	109
6.3	Black-Box Testing	110
6.3.1	Specification-Based Testing	111
6.3.2	Random Testing	116
6.3.3	Exposing Security Flaws with Fuzz-Testing	123
6.4	Key Points	124
7	Software Inspections, Code Reviews, and Safety Arguments	127
7.1	Formal Inspections	128
7.2	Modern Code Reviews - Reviewing Code During Development	128
7.2.1	Tool-Driven Code Review	129

- 7.2.2 Pull-Based Development 130
 - 7.2.3 The Impact of MCR on Software Development and Quality . 131
 - 7.3 Code Reviewing Techniques 132
 - 7.3.1 Tool-Driven Code Review 133
 - 7.3.2 Developer-driven Code Reviews 134
 - 7.4 Safety Arguments and Inspections of Safety Requirements 136
 - 7.4.1 Checklists 136
 - 7.4.2 Safety Argumentation and the Goal Structure Notation 138
 - 7.5 Key Points 139
- 8 Measurement 141**
 - 8.1 Measurement Basics 142
 - 8.2 Metrics 147
 - 8.2.1 Size and Complexity 148
 - 8.2.2 Modularity Metrics 153
 - 8.2.3 Maintainability Metrics and the Maintainability Index 158
 - 8.3 Validity and the Use of Goal Question Metric 159
 - 8.3.1 Problems of Validity 159
 - 8.3.2 Goal Question Metric 160
 - 8.4 Key Points 162
- 9 Conclusions 165**
 - 9.1 Topical and Emerging Quality Concerns 165
 - 9.1.1 Autonomy in Socio-Technical Systems 165
 - 9.1.2 Data-Intensive, Untestable Systems 167
 - 9.2 Concluding Remarks 169
- References 171**
- Index 179**