

Undergraduate Topics in Computer Science

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.

Also in this series

Iain D. Craig

Object-Oriented Programming Languages: Interpretation

978-1-84628-773-2

Max Bramer

Principles of Data Mining

978-1-84628-765-7

Hanne Riis Nielson and Flemming Nielson

Semantics with Applications: An Appetizer

978-1-84628-691-9

Michael Kifer and Scott A. Smolka

Introduction to Operating System Design and Implementation: The OSP 2 Approach

978-1-84628-842-5

Phil Brooke and Richard Paige

Practical Distributed Processing

978-1-84628-840-1

Frank Klawonn

Computer Graphics with Java

978-1-84628-847-0

David Salomon

A Concise Introduction to Data Compression

978-1-84800-071-1

David Makinson

Sets, Logic and Maths for Computing

978-1-84628-844-9

Alan P. Parkes

A Concise Introduction to Languages and Machines

978-1-84800-120-6

Orit Hazzan and Yael Dubinsky

Agile Software Engineering

 Springer

Orit Hazzan, BSc, MSc, PhD, MBA
Department of Education in Technology
and Science
Technion – Israel Institute of Technology
Haifa, Israel

Yael Dubinsky, BSc, MSc, PhD
Department of Computer Science
Technion – Israel Institute
of Technology
Haifa, Israel

Series editor

Ian Mackie, École Polytechnique, France and University of Sussex, UK

Advisory board

Samson Abramsky, University of Oxford, UK
Chris Hankin, Imperial College London, UK
Dexter Kozen, Cornell University, USA
Andrew Pitts, University of Cambridge, UK
Hanne Riis Nielson, Technical University of Denmark, Denmark
Steven Skiena, Stony Brook University, USA
Iain Stewart, University of Durham, UK
David Zhang, The Hong Kong Polytechnic University, Hong Kong

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN: 978-1-84800-198-5 e-ISBN: 978-1-84800-199-2
DOI: 10.1007/978-1-84800-199-2

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2008933449

© Springer-Verlag London Limited 2008

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

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

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

Springer Science+Business Media
springer.com

*Orit Hazzan dedicates this book to her agile family—
Shimon, Yael and Dan*

*Yael Dubinsky dedicates this book to her parents, Helen and Yaakov,
life partner David, and lovely children Tal and Or*

Preface

Overview and Goals

The agile approach for software development has been applied more and more extensively since the mid nineties of the 20th century. Though there are only about ten years of accumulated experience using the agile approach, it is currently conceived as one of the mainstream approaches for software development.

This book presents a complete software engineering course from the agile angle. Our intention is to present the agile approach in a holistic and comprehensive learning environment that fits both industry and academia and inspires the spirit of agile software development.

Agile software engineering is reviewed in this book through the following three perspectives:

- The **H**uman perspective, which includes cognitive and social aspects, and refers to learning and interpersonal processes between teammates, customers, and management.
- The **O**rganizational perspective, which includes managerial and cultural aspects, and refers to software project management and control.
- The **T**echnological perspective, which includes practical and technical aspects, and refers to design, testing, and coding, as well as to integration, delivery, and maintenance of software products.

Specifically, we explain and analyze how the explicit attention that agile software development gives these perspectives and their interconnections, helps

it cope with the challenges of software projects. This multifaceted perspective on software development processes is reflected in this book, among other ways, by the chapter titles, which specify dimensions of software development projects such as quality, time, abstraction, and management, rather than specific project stages, phases, or practices.


HOT


HOT


HOT

To share with the readers this multifaceted perspective, we use the **H**uman, **O**rganizational, and **T**echnical (HOT) scale for software development approaches. For example, when we refer to teamwork or abstraction levels, we emphasize the Human perspective; when software management issues are addressed, the Organizational perspective is emphasized; similarly, when the actual performance of test-driven development is described, the Technological aspect is highlighted. When the HOT? sign appears, the readers are invited to suggest their own HOT perspective.

Agile software development values these three perspectives. Therefore, in many cases, more than one perspective is illuminated by the agile approach with respect to a specific topic. Yet even when more than one perspective is significant with respect to a specific topic, we discuss from time to time only one or two main perspective(s), and the readers are invited to complete the picture.

The book is based on the authors' comprehensive experience of teaching and implementing agile software development over the past six years. A course on agile software engineering has been shaped during these years, in an iterative process that was accompanied by an ongoing research project. This course is presented in this book. In parallel to the course creation and shaping process, the agile approach has emerged and spread, becoming one of the worldwide mainstream approaches for software project management.

Organization and Features

This textbook guides a fourteen-week/session course on software engineering from the agile perspective and can be used on a weekly basis. It is intended for all who practice, research, teach, and learn software development both in academia and industry. It discusses how agile teams live and function in software development environments, how they achieve their goals, and how they act professionally in their environments. Specifically, the themes presented in the book, such as teamwork, time, quality, learning, trust, and culture, are reviewed from human, organizational, and technological perspectives, at the individual, team, and organizational levels, and are illustrated with case studies taken from industry and academia.

The fourteen chapters of the book are organized in three iterations. This structure enables us to revisit the various subjects several times during the course,

Table 1. Book structure

Iteration	Chapter #	Topic
I	1	Introduction to Agile Software Development
	2	Teamwork
	3	Customers and Users
	4	Time
	5	Measures
	6	Quality
	7	Learning
II	8	Abstraction
	9	Trust
	10	Globalization
	11	Reflection
III	12	Change
	13	Leadership
	14	Delivery and Cyclicity

as well as to guide the development of a one-release software product. Table 1 presents the book’s structure book and the topic of each chapter.

Each chapter includes a theoretical approach to a specific topic, a section that refers to the given topic in learning environments, and a variety of questions and tasks for further elaboration.

The Academic Community

This book on agile software engineering can be used by instructors, academic coaches, and students as a textbook during a fourteen-week semester, either for the commonly titled “Introduction to Software Engineering” course or the “Software Engineering Methods” course.

The course is based on two main components that progress in parallel and are closely correlated with each other. The first component is theoretical and can be used in the lecture hall or the class; the second is software project development guided by the agile approach that takes place in a physical learning environment that we call a studio or lab.

This book is written for the entire course community—students, instructors, and academic coaches. Students are the learners who become familiar with the agile approach both from a theoretical perspective (in the lectures) and from a practical perspective (in the studio). Instructors are the teachers of the course’s theoretical ideas, who usually teach in a class or in a lecture hall; yet, interactive teaching and active learning can be facilitated in this setting as well. The academic coaches are the practitioners who guide the software project development

in the studio (we elaborate on this role in Chapter 1, Introduction to Agile Software Engineering).

The positive results of agile software projects, as elaborated throughout the various chapters of the book, are not the only motive for this course, which presents the field of software engineering from the agile perspective. There are three additional characteristics of the course, which are especially relevant when it is taught in academia.

First, the agile approach was developed by practitioners working in the software industry, and has become mainstream in that industry. Therefore, it makes sense to articulate its nature and main concepts to prospective software engineers in the framework of a course that deals with software engineering.

Second, teaching a software engineering course within the framework of agile software development emphasizes a comprehensive image of the field. This is because agile software development explicitly addresses human, organizational, and technological aspects of the software development process with respect to all players participating in that process. Thus, the agile approach serves as an opportunity to draw this comprehensive and complex picture of the field.

Third, according to the Software Engineering 2004 Curriculum, developed by the IEEE Computer Society and the Association for Computing Machinery Joint Task Force (see <http://sites.computer.org/ccse/SE2004Volume.pdf>), software engineering students should acquire additional skills beyond the technical and scientific ones. One illustrative example is teamwork-related skills. Since teamwork is one of the basic ideas of agile software development, it is only natural to integrate teamwork-oriented skills in the teaching and learning process of software engineering from the agile perspective. Furthermore, since it is natural to teach agile software development in a teamwork-oriented environment, there is no need to introduce the topic of teamwork artificially; rather, a teamwork-based learning environment can be used to teach this topic. This element is emphasized mainly, but not only, in the studio element of the course.

Suggested Uses in an Academic Environment

Each chapter presents a full week of the course: two weekly lecture hours and a four-hour weekly studio meeting. The first part of each chapter includes contents suitable to be presented in the lecture. This part usually presents material beyond what it is possible to teach in a two-hour lecture. Therefore, it is advisable not to try to deliver all the content in two hours; rather, we suggest selecting from each chapter the most relevant topics to be discussed with each particular class of students. It is also advisable to encourage in the lectures some active learning elements, as is suggested in the various chapters. The second part of each chapter

addresses the teaching and learning of the chapter topic. It presents teaching and learning principles and the activities conducted in the studio each week.

As preparation for the next week's lectures and studio meeting, instructors and academic coaches can ask the students to read the relevant chapter and to work on selected activities presented throughout the body of each chapter. The students' preparation for the lecture will also partially solve the time limitation problem of addressing all the ideas presented.

Finally, though the book presents a full fourteen-week semester course, which consists of two weekly lecture hours and four-hour weekly studio meetings, it is possible to teach only one component of the course. The material provided in this book enables each instructor/academic coach to make the needed adjustments.

The Industrial Community

Since agile development has become one of the mainstream approaches for managing software projects, more and more software organizations of different sizes and types ask themselves whether the agile approach fits them. Even when it is found that agile software development is relevant for a given organization, questions such as the following are usually asked: How can we manage a transition to the agile software development process? How can our organization cope with the changes required for such a transition? How can we teach agile software development to all the software practitioners and all the other software project's stakeholders?

This book, when used in an industrial setting, aims to answer these and other relevant questions which software organizations face when dealing with the transition to agile software development. For example, in Chapter 12, Change, we discuss how to initiate a transition process to agile software development in an organization. When the organization has already transitioned to agile software development, the book can also be used for answering questions related to the actual implementation of agile software development in the organization. For example, in Chapter 2, Teamwork, we discuss how teams can be formed to exploit their potential, to avoid conflicts, and to solve dilemmas.

Suggested Uses in an Industrial Environment

This book can be used in industrial settings by coaches of software teams, software team leaders, and facilitators of agile software development workshops, both for the teaching and learning of agile software development, as well as for its implementation. The book can also be used by interested software practitioners who are not necessarily within a formal teaching framework.

We propose two ways to use the book in industrial environments.

First, the book can be used for a course which is based on 14 sessions. This course format fits for organizations that wish to expand their members' professional knowledge by becoming familiar with agile software development, without necessarily implementing the agile approach. If the course also contains the development of a software project using the agile approach, which in academia takes place in the studio, a new software system should be developed for learning process purposes, with respect to which the different activities are facilitated. The development of a new software project should be undertaken whether the course is taught to a real team or to a group of people from different teams or organization. In the case of a real team, the development of another project than the team's real project will enable the team not to confuse their current work habits with agile practices.

Second, for organizations which wish to start implementing agile software development right away or in the near future, we suggest that the agile approach be taught first in a short format of a two-day workshop to a team that has been carefully selected to start the transition to agile software development within the organization. Chapter 12, Change, elaborates on such a transition process, explains the motivation and rationale for this intense workshop format, and outlines the workshop schedule. After the team members have participated in that workshop, and when the team starts implementing agile software development with its real project, the book can be used for clarifications and elaborations.

In both cases, as well as in other learning environments in industry, the teaching and learning principles presented in the book can naturally be applied.

Acknowledgments

We would like to thank all the practitioners, researchers, students, and managers, both in academia and in the software industry, who during the past six years shared with us their professional knowledge, experience, thoughts, and feelings with respect to agile software development. They all contributed to our understanding of the nature of agile software engineering and fostered our shaping of the approach presented in this book.

Contents

1. Introduction to Agile Software Development	1
1.1 Overview	1
1.2 Objectives	2
1.3 Study Questions	2
1.4 Three Perspectives on Software Engineering	3
1.5 The Agile Manifesto	4
1.5.1 Individuals and Interactions over Processes and Tools . .	5
1.5.2 Working Software over Comprehensive Documentation	6
1.5.3 Customer Collaboration over Contract Negotiation . . .	7
1.5.4 Responding to Change over Following a Plan	7
1.6 Application of Agile Software Development	8
1.7 Data About Agile Software Development	13
1.8 Agile Software Development in Learning Environments	15
1.8.1 University Course Structure	15
1.8.2 Teaching and Learning Principles	15
1.8.3 The Studio Environment	17
1.8.4 The Academic Coach Role	18
1.8.5 Overview of the Studio Meetings	19
1.8.6 Launching the Project Development in the Studio	20
1.9 Summary and Reflective Questions	23
1.10 Summary	24
References	24
2. Teamwork	25
2.1 Overview	25
2.2 Objectives	26
2.3 Study Questions	26
2.4 A Role Scheme in Agile Teams	27
2.4.1 Remarks on the Implementation of the Role Scheme . . .	31
2.4.2 Human Perspective on the Role Scheme	32
2.4.3 Using the Role Scheme to Scale Agile Projects	34
2.5 Dilemmas in Teamwork	34
2.6 Teamwork in Learning Environments	36

2.6.1	Teaching and Learning Principles	36
2.6.2	Role Activities	37
2.6.3	Student Evaluation	40
2.7	Concluding Reflective Questions	42
2.8	Summary	42
	References	42
3.	Customers and Users	45
3.1	Overview	45
3.2	Objectives	47
3.3	Study Questions	47
3.4	The Customer	48
3.4.1	Customer Role	48
3.4.2	Customer Collaboration	54
3.5	The User	55
3.5.1	Combining UCD with Agile Development	57
3.6	Customers and Users in Learning Environments	61
3.6.1	Teaching and Learning Principles	61
3.6.2	Customer Stories	62
3.6.3	Case Studies of Metaphor Use	62
3.7	Summary and Reflective Questions	67
3.8	Summary	68
	References	68
4.	Time	71
4.1	Overview	71
4.2	Objectives	72
4.3	Study Questions	72
4.4	Time-Related Problems in Software Projects	73
4.4.1	List of Time-Related Problems of Software Projects	74
4.4.2	Case Study 4.1. Software Organizational Survey from the Time Perspective	75
4.5	Tightness of Software Development Methods	77
4.6	Sustainable Pace	79
4.6.1	Case Study 4.2. An Iteration Timetable of an Agile Team	80
4.7	Time Management of Agile Projects	81
4.7.1	Time Measurements	81
4.7.2	Prioritizing Development Tasks	83
4.8	Time in Learning Environments	86
4.8.1	The Planning Activity	86
4.8.2	Teaching and Learning Principles	88
4.8.3	Students' Reflections on Time-Related Issues	89
4.8.4	The Academic Coach's Perspective	89

4.9	Summary and Reflective Questions	90
4.10	Summary	91
	References	91
5.	Measures	93
5.1	Overview	93
5.2	Objectives	95
5.3	Study Questions	95
5.4	Why Are Measures Needed?	95
5.5	Who Decides What Is Measured?	96
5.6	What Should Be Measured?	97
5.7	When Are Measures Taken?	98
5.8	How Are Measures Taken?	98
5.9	Who Takes the Measures?	99
5.10	How Are Measures Used?	99
5.11	Case Study 5.1. Monitoring a Large-Scale Project by Measures	100
	5.11.1 Measure Definition	100
	5.11.2 Measure Illustration	102
5.12	Measures in Learning Environments	108
	5.12.1 Teaching and Learning Principles	108
	5.12.2 Measurement Activities	109
	5.12.3 Case Study 5.2. Role-Related Measures	111
5.13	Summary and Reflective Questions	114
5.14	Summary	114
	References	114
6.	Quality	115
6.1	Overview	115
6.2	Objectives	116
6.3	Study Questions	117
6.4	The Agile Approach to Quality Assurance	117
	6.4.1 Process Quality	119
	6.4.2 Product Quality	120
6.5	Test-Driven Development	121
	6.5.1 How Does TDD Help Overcome Some of the Problems Inherent in Testing?	122
	6.5.2 Case Study 6.1. TDD Steps	124
	6.5.3 Case Study 6.2. Reflection on TDD	125
6.6	Measured TDD	127
6.7	Quality in Learning Environments	128
	6.7.1 Case Study 6.3. Size and Complexity Measures	128

6.7.2	Case Study 6.4. Illustrating Measured TDD.	130
6.7.3	Teaching and Learning Principles—The Case of Quality.	136
6.8	Summary and Reflective Questions	137
6.9	Summary.	137
	References	138
7.	Learning	139
7.1	Overview.	139
7.2	Objectives	140
7.3	Study Questions	140
7.4	How Does Agile Software Development Support Learning Processes?	141
7.4.1	Agile Software Development from the Constructivist Perspective.	141
7.4.2	The Role of Short Releases and Iterations in Learning Processes	142
7.5	Learning in Learning Environments.	144
7.5.1	Gradual Learning Process of Agile Software Engineering	145
7.5.2	Learning and Teaching Principle	146
7.5.3	The Studio Meeting—End of the First Iteration	147
7.5.4	Intermediate Course Review and Reflection.	147
7.6	Summary and Reflective Questions	152
7.7	Summary.	152
	References	152
8.	Abstraction	155
8.1	Overview.	155
8.2	Objectives	156
8.3	Study Questions	157
8.4	Abstraction Levels in Agile Software Development.	158
8.4.1	Roles in Agile Teams.	158
8.4.2	Case Study 8.1. Abstraction During Iteration Planning	159
8.4.3	The Stand-Up Meeting	161
8.4.4	Design and Refactoring	162
8.5	Abstraction in Learning Environments	164
8.5.1	Teaching and Learning Principles.	165
8.5.2	Case Study 8.2. RefactoringActivity.	166
8.6	Summary and Reflective Questions	169
8.7	Summary.	170
	References	170

9. Trust	171
9.1 Overview	171
9.2 Objectives	172
9.3 Study Questions	172
9.4 Software Intangibility and Process Transparency	173
9.5 Game Theory Perspective in Software Development	175
9.6 Ethics in Agile Teams	179
9.7 Diversity	183
9.8 Trust in Learning Environments	186
9.8.1 Teaching and Learning Principle	186
9.9 Summary and Reflective Questions	187
9.10 Summary	188
References	188
10. Globalization	189
10.1 Overview	190
10.2 Objectives	190
10.3 Study Questions	191
10.4 The Agile Approach in Global Software Development	191
10.4.1 Communication in Distributed Agile Teams	192
10.4.2 Planning in Distributed Agile Projects	193
10.4.3 Case Study 10.1. Tracking Agile Distributed Projects	193
10.4.4 Reflective Processes in Agile Distributed Teams	194
10.4.5 Organizational Culture and Agile Distributed Teams	195
10.5 Application of Agile Principles in Non-Software Projects	196
10.5.1 Case Study 10.2. Book Writing	196
10.6 Globalization in Learning Environments	197
10.6.1 Teaching and Learning Principles	197
10.6.2 An Agile Perspective on the Book/Course Structure	198
10.6.3 Case Study 10.3. Follow-the-Sun with Agile Development	199
10.7 Summary and Reflective Questions	201
10.8 Summary	202
References	202
11. Reflection	205
11.1 Overview	205
11.2 Objectives	206
11.3 Study Questions	206
11.4 Case Study 11.1. Reflection on Learning in Agile Software Development	207

11.5	Reflective Practitioner Perspective	208
11.6	Retrospective	210
11.6.1	The Retrospective Facilitator	211
11.6.2	Case Study 11.2. Guidelines for a Retrospective Session	212
11.6.3	Application of Agile Practices in Retrospective Sessions	213
11.6.4	End of the Release Retrospective	215
11.7	Reflection in Learning Environments	219
11.8	Summary and Reflective Questions	219
11.9	Summary	220
	References	220
12.	Change	223
12.1	Overview	223
12.2	Objectives	224
12.3	Study Questions	225
12.4	A Conceptual Framework for Change Introduction	225
12.4.1	Changes in Software Requirements	227
12.4.2	Organizational Changes	230
12.5	Transition to an Agile Software Development Environment	234
12.5.1	Organizational Survey	235
12.5.2	Case Study 12.1. A Report of an Organizational Survey	237
12.5.3	Case Study 12.2. Applying an Agile Process to a Transition Process	241
12.6	Change in Learning Environments	244
12.6.1	Introducing the Teaching of Agile Software Development	244
12.6.2	Two-Day Workshop	245
12.6.3	Two-Day Workshop Format for a Team of Academic Coaches	250
12.7	Summary and Reflective Questions	251
12.8	Summary	252
	References	252
13.	Leadership	253
13.1	Overview	253
13.2	Objectives	255
13.3	Study Questions	255
13.4	Leaders	256
13.4.1	Leadership Styles	257
13.4.2	Case Study 13.1. The Agile Change Leader	258
13.5	Coaches	264

13.6	Leadership in Learning Environments	264
13.6.1	Teaching and Learning Principles	265
13.6.2	Case Study 13.2. A Coaching Framework	265
13.7	Summary and Reflective Questions	273
13.8	Summary	273
	References	273
14.	Delivery and Cyclicity	275
14.1	Overview	275
14.2	Objectives	276
14.3	Study Questions	276
14.4	Delivery	277
14.4.1	Towards the End of the Release	277
14.4.2	Release Celebration	278
14.4.3	Reflective Session Between Releases	280
14.5	Cyclicity	287
14.6	Delivery and Cyclicity in Learning Environments	288
14.6.1	The Delivery in the Studio	288
14.6.2	Teaching and Learning Principles	290
14.7	Summary and Reflective Questions	291
14.8	Summary	291
	References	292
	Epilogue	293
	Index	295