Undergraduate Topics in Computer Science (UTiCS) delivers high-quality

instructional content for under- graduates studying in all areas of computing and information science. From core foundational and theoreti- cal 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.

Department of Computer Science and Engineering IIT Delhi, India

Ian Mackie, E´ cole Polytechnique, France and University of Sussex, UK

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

David Zhang, The Hong Kong Polytechnic University, Hong Kong

Undergraduate Topics in Computer Science ISSN: 1863-7310 ISBN: 978-1-84800-301-9 e-ISBN: 978-1-84800-302-: 10.1007/978-1-84800-302-6

A catalogue record for this book is available from the British Library Library of Congress Control Number: 2008933221

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 licenses 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.

An introductory course on Software Engineering remains one of the hardest subjects to teach largely because of the wide range of topics the area encom- passes. I have believed for some time that we often tend to teach too many concepts and topics in an introductory course resulting in shallow knowledge and little insight on application of these concepts. And Software Engineering is finally about application of concepts to efficiently engineer good software solutions.

I believe that an introductory course on Software Engineering should focus on imparting to students the knowledge and skills that are needed to successfully execute a commercial project of a few person-months effort while employing proper practices and techniques. It is worth pointing out that a vast majority of the projects executed in the industry today fall in this scope—executed by a small team over a few months. I also believe that by carefully selecting the concepts and topics, we can, in the course of a semester, achieve this. This is the motivation of this book.

The goal of this book is to introduce to the students a limited number of concepts and practices which will achieve the following two objectives:

Teach the student the skills needed to execute a smallish commercial project.

Provide the students necessary conceptual background for undertaking ad- vanced studies in software engineering, through courses or on their own.

I have included in this book only those concepts that I believe are founda- tional and through which the two objectives mentioned above can be met. Ad- vanced topics have been consciously left out. As executing a software project requires skills in two dimensions—engineering and project management—this book focuses on key tasks in these two dimensions, and discusses concepts and techniques that can be applied to effectively execute these tasks.

The book is organized in a simple manner, with one chapter for each of the key tasks in a project. For engineering, these tasks are requirements analy- sis and specification, architecture design, module level design, coding and unit testing, and testing. For project management, the key tasks are project plan- ning and project monitoring and control, but both are discussed together in one chapter on project planning as even monitoring has to be planned. In addi- tion, the book contains one chapter that clearly defines the problem domain of Software Engineering, and another chapter that discusses the central concept of software process which integrates the different tasks executed in a project.

Each chapter opens with some introduction and then clearly lists the chapter goals, or what the reader can expect to learn from the chapter. For the task covered in the chapter, the important concepts are first discussed, followed by a discussion of the output of the task, the desired quality properties of the output, and some practical methods and notations for performing the task. The explanations are supported by examples, and the key learnings are summarized in the end for the reader. The chapter ends with some self-assessment exercises.

The book is primarily intented for an introductory course on Software Engi- neering in any undergraduate or graduate program. It is targeted for students who know programming but have not had a formal exposure to software engi- neering.

The book can also be used by professionals who are in a similar state—know some programming but want to be introduced to the systematic approach of software engineering.

Though the book is self-contained, some teaching support and supplemental resources are available through a website. The URL is:

The resources available on the site include:

The powerpoint presentations for each chapter in ppt format so instructors can change them to suit their style.

Various templates for different outputs in a project, that can be used for the student project in the course.

A case study with most of the major outputs of the project.

Some practice exercises for unit testing and inspections.

I would like to express my gratitude to my editor, Wayne Wheeler, who con- ceived this idea of a concise introductory book and created this opportunity.

I would also like to express my thanks to my wife, Shikha, and my daughters Sumedha and Sunanda for once again bearing with my moods and odd hours.

Ask any student who has had some programming experience the following question: You are given a problem for which you have to build a software system that most students feel will be approximately 10,000 lines of (say C or Java) code. If you are working full time on it, how long will it take you to build this system?

The answer of students is generally . And, given the program- ming expertise of the students, there is a good chance that they will be able to build the software and demo it to the professor within . With , the productivity of the student will be 5000 lines of code (LOC) per person-month.

Now let us take an alternative scenario—we act as clients and pose the same problem to a company that is in the business of developing software for clients. Though there is no standard productivity figure and it varies a lot, it is fair to say a productivity figure of 1000 LOC per person-month is quite respectable (though it can be as low as 100 LOC per person-month for embedded systems). With this productivity, a team of professionals in a software organization will take 10 person-months to build this software system.

Why this difference in productivity in the two scenarios? Why is it that the same students who can produce software at a productivity of a few thousand LOC per month while in college end up producing only about a thousand LOC per month when working in a company?