Preface

This volume contains the lecture notes of the courses given at the School on Engineering Trustworthy Software Systems (SETSS), held during September 8–13, 2014, at Southwest University in Chongqing, China. The school was aimed at post-graduate students, researchers, academics, and industrial engineers who are interested in the theory and practice of methods and tools for the design and programming of trustworthy software systems.

It is widely known that software engineering aims to develop and study theoretical foundations and practical disciplines for software design and production, ones that are as effective as as those already established in traditional branches of engineering. Yet although formal theories of programming, techniques, and tools do exist, already developed with their underpinnings, they are in fact not widely practised by software engineering practitioners; and so the impact of formal methods on commonly used software systems is still far from convincing. Indeed, it is not widely understood where and how the practices of software engineering are informed, and by what theories. Here we quote the question that Carroll Morgan raised in his first lecture at the school:

Trustworthy Systems: *Would you trust...*

A mechanical engineer who did not understand torque?	$\frac{\mathrm{d}(r\times\omega)}{\mathrm{d}t} = r \times \frac{\mathrm{d}\omega}{\mathrm{d}t} + \frac{\mathrm{d}r}{\mathrm{d}t} \times \omega$
An electrical engineer who did not understand impedance?	$e^{j\omega t} = \cos(\omega t) + j\sin(\omega t)$
A civil engineer who did not understand trigonometry?	$\sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi$
⋮	
A software engineer *who did not understand assertions?*	$\{post[x\backslash E]\}\ x := E\ \{post\}$

Carroll Morgan, SETSS 2014
Chongqing, China, 8 September 2014

Carroll continued to discuss in the class how a traditional engineer learns the above fundamental theories "in the first/second year of undergraduate university studies, or even in high school," but that the situation for a software engineer is quite different.

The courses of SETSS 2014 aimed to improve the understanding of the relation between theory and practice in software engineering, in order to contribute to narrowing the gap between them. This volume contains the lecture notes of the five courses and materials of one seminar. The common themes of the courses include the design and use of theories, techniques, and tools for software specification and

modeling, analysis, and verification. The courses cover sequential programming, component and object software, hybrid systems, and cyber-physical systems with challenges of termination, security, safety, fault-tolerance, and real-time requirements. The techniques include model checking, correctness by construction through refinement and model transformations, as well as synthesis and computer algebra.

Lecturers and Editors

JONATHAN P. BOWEN was Professor of Computer Science in the Faculty of Computing, Engineering, and the Built Environment and Deputy Head of the Centre for Software Engineering of Birmingham City University (UK) between 2013 and 2015. He has been Emeritus Professor of Computing at London South Bank University since 2007. During 2008–2009, Bowen worked on a major air traffic control project at Altran Praxis (now Altran UK), applying the formal Z notation to a real industrial application. Bowen originally studied engineering science at Oxford University. He has been working in the field of computer science since the late 1970s, mainly in academia but also in industry. His previous academic affiliations include Imperial College (London), the Oxford University Computing Laboratory, and the University of Reading, as well as a number of visiting positions internationally, most recently at the Israel Institute for Advanced Studies during 2015–2016. Bowen is Life Fellow of the British Computer Society and the Royal Society of Arts. His professional interests include formal methods, software engineering, and museum informatics.

ZHIMING LIU just joined Southwest University (Chongqing, China) as a professor, leading the development of the Centre for Software Research and Innovation (SIRC). Before that, he was Professor of Software Engineering and Head of the Centre for Software Engineering at Birmingham City University (UK, 2013–2015) as well as Senior Research Fellow of the United Nations University – International Institute for Software Technology (Macao, 2002-1013). His is known for his work on the transformational approach to real-time and fault-tolerant system specification and verification, and the rCOS formal-model-driven-software engineering method.

ANNABELLE McIVER is a professor in the Department of Computing at Macquarie University in Sydney, where she is also Director of Research. She was trained as a mathematician at Cambridge and Oxford universities and in her research she uses mathematics to analyze security flaws in computer systems.

She is a member of the Programming Methodology Technical Working Group of the International Federation of Information Processing.

CARROLL MORGAN is a professor at the University of New South Wales and Data61 (formerly NICTA). He is known for his work, with his colleagues, on formal methods generally: originally Z, then refinement calculus, then probabilistic weakest-preconditions, and most recently “The Shadow” model for abstraction and refinement of non-interference security properties. These last two together are combined in his current work on quantitative information flow.

He also has a keen interest on how formal methods can be taught to beginner programmers… before it is too late.

BERND-HOLGER SCHLINGLOFF is a professor for software technology at the Humboldt University of Berlin, with a research focus on specification, verification, and testing theory of embedded systems. At the same time, he is a chief scientist at the Fraunhofer Institute of Open Communication Systems FOKUS in Berlin. His research interests are in the quality assurance of cyber-physical systems, in particular, in the automated generation and execution of software tests with formal specifications. Prof. Schlingloff is an internationally acknowledged expert in this field and has published more than 20 scientific articles and book chapters on this subject within the last five years. He coordinates several European and national projects in these areas, and uses the results in industrial projects within the domains of railway, traffic, automation, and medicine.

NAIJUN ZHAN is a distinguished professor of the Chinese Academy of Sciences, Deputy Director of the State Key Laboratory of Computer Science at the Institute of Software of the Academy, and a professor of the University of the Chinese Academy of Sciences. He is known for his work on formal design of real-time systems, and, in particular, his work on formal verification of hybrid systems.

ZILI ZHANG is Professor and Dean of the Faculty of Computer and Information Sciences at Southwest University. He is also a senior lecturer in the School of Information Technology at Deakin University. He has over 130 refereed publications in journal and conferences, one monograph, and six textbooks. He has been awarded about 30 grants from both China and Australia. His research interests include bio-inspired artificial intelligence, agent-based computing, big data analysis, and agent–data mining interaction and integration. He is a member of the Chinese Computer Federation (CCF) Big Data Task Force, the chief expert of the Chongqing Agricultural and Rural Digitalization Program, and a member of the Expert Committee for the Chongqing Cloud Computing Program.

Lecture Courses

Course 1: (In-)Formal Methods: The Lost Art — A Users' Manual by Carroll Morgan. The course draws from an experimental course in "(In-)Formal Methods," taught for three years at the University of New South Wales to fourth-year undergraduate computer science students. An adapted version was then taught (disguised as "Software Engineering") to second-year undergraduate students. The purpose is to "lower the barrier" for the use of those techniques, to show how astonishingly useful they are even without using propositional calculus, or predicate calculus — even if you figure out your invariants using pictures or by waving your hands in the air. Thus even students who have heard of Hoare triples might benefit from this course if they have not actually used them. The material is supported by the use of the program-correctness prover Dafny: students will see how to design correctness arguments, develop programs guided by those arguments, and then finally submit the arguments and the programs together for automated checking. This volume is divided into two parts, Part I on the generalities and Part II on the specifics.

Course 2: Program Refinement, Perfect Secrecy, and Information Flow by Annabelle McIver. This course is about a method for security by construction, which

extends traditional "programming by stepwise refinement" as described in Course 1. This "comparative approach" features in stepwise refinement: describe a system as simply as possible so that it has exactly the required properties, and then apply sound refinement rules to obtain an implementation comprising specific algorithms and data structures. The stepwise refinement method has been extended to include "information flow" properties as well as functional properties, thus supporting proofs about secrecy within the program refinement method. In this course, the security-by-refinement approach is reviewed and it is illustrated how it can be used to give an elementary treatment of some well-known security principles.

Course 3: The Z Notation – Whence the Cause and Whither the Course? by Jonathan P. Bowen.

This is a course on the Z notation for the formal specification of computer-based systems that has been in existence since the early 1980s. Since then, an international Z community has emerged, academic and industrial courses have been developed, an ISO standard has been adopted, and Z has been used on a number of significant software development projects, especially where safety and security have been important. This chapter traces the history of the Z notation and presents issues in teaching Z, with examples. A specific example of an industrial course is presented. Although subsequent notations have been developed, with better tool support, Z is still an excellent choice for general-purpose specification and is especially useful in directing software testing to ensure good coverage.

Course 4: Model-Driven Design of Object Component Systems by Zhiming Liu. This course identifies a set of UML notations and textual descriptions for representing different abstractions of software artifacts produced in different development stages. These abstractions, their relations and manipulations all have formal definitions in the rCOS formal method of component and object systems. The purpose is to show how model-driven development seamlessly integrates the theories that have been well developed in the last half century, including abstract data types, Hoare logic, process calculi, I/O automata as well as their underlined techniques and tools for software specification, refinement, and verification. A major theme is to show that models of component-based architectures and interface contracts are essential for designing and maintaining large-scale evolving systems, including cyber-physical systems (CPS), Internet of Things (IoT), and Smart Cities, which have multi-dimensional complexities. The lecture notes in this volume are divided into three consecutive parts:

- Part I describes the background motivation and organization. It especially gives a historic account of software engineering, which is usually missing in textbooks and lecture notes, and discusses basic concepts and principles of model-driven software design.
- Part II is devoted to use-case-driven object-oriented requirements gathering, modeling and analysis, and the UML models used for representing requirement artifacts.
- Part III covers component-based architecture design, and object-oriented design architectural components based on their contracts of interfaces.

Course 5: Cyber-Physical Systems Engineering by Holger Schlingloff. Cyber-physical systems, that is, connected devices that support technical processes and human

users, have become ubiquitous in our environment. Examples range from connected AV home entertainment and smart home automation systems, via intelligent cars, UAVs, and autonomous robots, to fully automated factories. However, the complexity of these systems is steadily growing, making it increasingly harder to design them correctly.

Building complex embedded and cyber-physical systems requires a holistic view. Systems engineering must provide means to continuously consider both the design process and the final product:

- The development processes must provide a seamless transition between different stages and views.
- The constructed system must offer a smooth interaction with its physical environment and its human users.

Thus, for cyber-physical systems engineering, modeling techniques and methods have been developed that support such an integral design paradigm.

In this course the fundamentals of cyber-physical systems engineering are presented: identification and quantification of system goals; requirements elicitation and management; modeling and simulation in different views; and validation to ensure that the system meets its original design goals. A special focus is on the model-based design process. All techniques are demonstrated with appropriate examples and engineering tools.

Course 6: Combining Formal and Informal Methods in the Design of Spacecrafts by Naijun Zhan. This course presents a combination of formal and informal methods for the design of spacecrafts. In the described approach, the designer can either build an executable model of a spacecraft using the industrial standard environment Simulink/Stateflow, or construct a formal model using Hybrid CSP, which is an extension of CSP for modeling hybrid systems. Hybrid CSP processes are specified and reasoned about by hybrid Hoare logic, which is an extension of Hoare logic to hybrid systems. The connection between informal and formal methods is realized via an automatic translator from Simulink/Stateflow diagrams to Hybrid CSP and an inverse translator from Hybrid CSP to Simulink. The course shows the following advantages of combining formal and informal methods in the design of spacecrafts:

- It allows formal verification to be used as a complement of simulation that, by itself, would be incomplete for system correctness.
- It allows the design of a hybrid system to be formally specified in Hybrid CSP and simulated and/or tested economically using the Matlab platform.

The method is demonstrated by analysis and verification of a real-world industry example, that is, the guidance control program of a lunar lander.

Acknowledgments. We would like to thank the lecturers and their co-authors for their professional commitment and hard work, the strong support of Southwest University, and the enthusiastic work of the local organizing team led by Dr. Li Tao, without which the school would not have been possible. We are grateful for the support of Alfred Hofmann and Anna Kramer of Springer's *Lecture Notes in Computer Science* team in the publication of this volume.

January 2016

Zhiming Liu
Zili Zhang

http://www.springer.com/978-3-319-29627-2

Engineering Trustworthy Software Systems
First International School, SETSS 2014, Chongqing, China, September 8-13, 2014. Tutorial Lectures
Liu, Z.; Zhang, Z. (Eds.)
2016, XI, 325 p. 141 illus. in color., Softcover
ISBN: 978-3-319-29627-2