Preface

Software has long been perceived as complex, at least within Software Engineering circles. We have been living in a recognised state of crisis since the first NATO Software Engineering conference in 1968. Time and again we have been proven unable to engineer software as easily/cheaply/safely as we imagined. Cost overruns and expensive failures are the norm.

The problem is fundamentally one of complexity—translating a problem specification into a form that can be solved by a computer is a complex undertaking. Any problem, no matter how well specified, will contain a baseline of intrinsic complexity—otherwise it is not much of a problem. Additional complexities accrue as a solution to the problem is implemented. As these increase, the complexity of the problem (and solution) quickly surpasses the ability of a single human to fully comprehend it. As team members are added new complexities will inevitably arise.

Software is fundamentally complex because it must be precise; errors will be ruthlessly punished by the computer. Problems that appear to be specified quite easily in plain language become far more complex when written in a more formal notation, such as computer code. Comparisons with other engineering disciplines are deceptive. One cannot easily increase the factor of safety of software in the same way that one could in building a steel structure, for example. Software is typically built assuming perfection, often without adequate safety nets in case the unthinkable happens. In such circumstances it should not be surprising to find out that (seemingly) minor errors have the potential to cause entire software systems to collapse. A worrying consideration is that the addition of additional safety or fault protection components to a system will also increase the system's overall complexity, potentially making the system *less safe*.

Our goal in this book is to uncover techniques that will aid in overcoming complexity and enable us to produce reliable, dependable computer systems that will operate as intended, and yet are produced on-time, in budget, and are evolvable, both over time and at run time. We hope that the contributions in this book will aid in understanding the nature of software complexity and provide guidance for the control or avoidance of complexity in the engineering of complex software systems. The book is organised into three parts: Part I (Chaps. 1 and 2) addresses the sources

and types of complexity; Part II (Chaps. 3 to 9) addresses areas of significance in dealing with complexity; Part III (Chaps. 10 to 17) identifies particular application areas and means of controlling complexity in those areas.

Part I of the book (Chaps. 1 and 2) drill down into the question of how to recognise and handle complexity. In tackling complexity two main tools are highlighted: abstraction and decomposition/composition. Throughout this book we see these tools reused, in different ways, to tackle the problem of *Controlling Complexity*.

In Chap. 1 José Luiz Fiadeiro discusses the nature of complexity and highlights the fact that software engineering seems to have been in a permanent state of crisis, a crisis might better be described as one of complexity. The difficulty we have in conquering it is that the nature of complexity itself is always changing. His sentiment that we cannot hope to do more than "shift [...] complexity to a place where it can be managed more effectively" is echoed throughout this book.

In Chap. 2 Michael Jackson outlines a number of different ways of decomposing system behaviour, based on the system's constituents, on machine events, on requirement events, use cases, or software modules. He highlights that although each offers advantages in different contexts, they are in themselves not adequate to master behavioural complexity. In addition he highlights the potential for oversimplification. If we decompose and isolate parts of the system and take into account only each part's intrinsic complexities we can easily miss some interactions between the systems, leading to potentially surprising system behaviour.

Part II of the book outlines different approaches to managing or controlling complexity. Chapters 3 and 4 discuss the need to tackle complexity in safety-critical systems, arguing that only by simplifying software can it be proven safe to use. These chapters argue for redundancy and separation of control and safety systems respectively.

Gerard Holzmann addresses the question of producing defect-free code in Chap. 3. He argues that rather than focusing on eliminating component failure by producing perfect systems, we should aim to minimise the possibility of system failure by focusing on the production of fallback redundant systems that are much simpler—simple enough to be verifiably correct. In Chap. 4, Wassyng et al. argue that rather than seeking to tame complexity we should focus our efforts on avoiding it altogether whenever reliability is paramount. The authors agree with Holzmann in that simpler systems are more easy to prove safe, but rather than using redundant systems to take control in the case of component failure they argue for the complete separation of systems that must be correct (in this case safety systems) from control systems.

In Chap. 5, Norman Schneidewind shows how it is possible to analyse the tradeoffs in a system between complexity, reliability, maintainability, and availability *prior to implementation*, which may reduce the uncertainty and highlight potential dangers in software evolution. In Chap. 6, Bohner et al argue that change tolerance must be built into the software and that accepting some complexity today to decrease the long term complexity that creeps in due to change is warranted.

Chapters 7 to 9 discuss autonomous, agent-based, and swarm-like software systems. The complexity that arises out of these systems comes from the interactions between the system's component actors or agents.

In Chap. 7 Hinchey et al. point out that new classes of systems are introducing new complexities, heretofore unseen in (mainstream) software engineering. They describe the complexities that arise when autonomous and autonomic characteristics are built into software, which are compounded when agents are enabled to interact with one another and self-organise. In Chap. 8 Mike Hinchey and Roy Sterritt discuss the techniques that have emerged from taking inspiration from biological systems. The autonomic nervous system has inspired approaches in autonomic computing, especially in self-managing, self-healing, and other self-* behaviours. They consider mechanisms that enable social insects (especially ants) to tackle problems as a colony (or "swarm" in the more general sense) and show how these can be applied to complex tasks. Peña et al. give a set of guidelines to show how complexity derived from interactions in agent-oriented software can be managed in Chap. 9. They use the example of the Ant Colony to model how complex goals can be achieved using small numbers of simple actors and their interactions with each other.

Part III of the book (Chaps. 10 to 17) discusses the control of complexity in different application areas. In Chap. 10, Tiziana Margaria and Bernhard Steffen argue that classical software development is no longer adequate for the bulk of application programming. Their goal is to manage the division of labour in order to minimise the complexity that is "felt" by each stakeholder.

The use of formal methods will always have a role when correct functioning of the software is critical. In Chap. 11, Jonathan Bowen and Mike Hinchey examine the attitudes towards formal methods in an attempt to answer the question as to why the software engineering community is not willing to either abandon or embrace formal methods. In Chap. 12 Filieri et al. focus on how to manage design-time uncertainty and run-time changes and how to verify that the software evolves dynamically without disrupting the reliability or performance of applications. In Chap. 13, Wei et al. present a timebands model that can explicitly recognise a finite set of distinct time bands in which temporal properties and associated behaviours are described. They demonstrate how significantly their model contributes to describing complex real-time systems with multiple time scales. In Chap. 14 Manfred Broy introduces a comprehensive theory for describing multifunctional software-intensive systems in terms of their interfaces, architectures and states. This supports the development of distributed systems with multifunctional behaviours and provides a number of structuring concepts for engineering larger, more complex systems.

In Chap. 15, John Anderson and Todd Carrico describe the Distributed Intelligent Agent Framework, which defines the essential elements of an agent-based system and its development/execution environment. This framework is useful for tackling the complexities of systems that consist of a large network of simple components without central control. Margaria et al. discuss the difficulties in dealing with monolithic ERP systems in Chap. 16. As the business needs of customers change the ERP system they use must change to respond to those needs. The requirements of flexibility and customisability introduce significant complexities, which much be overcome if the ERP providers are to remain competitive. In Chap. 17 Casanova et al. discuss the problem of matching database schemas. They introduce procedures

to test strict satisfiability and decide logical implication for extralite schemas with role hierarchies. These are sufficiently expressive to encode commonly-used Entity-Relationship model and UML constructs.

We would like to thank all authors for the work they put into their contributions. We would like to thank Springer for agreeing to publish this work and in particular Beverley Ford, for her support and encouragement. We would like to thank all of our friends and colleagues in Lero.[1]

Limerick, Ireland

Mike Hinchey
Lorcan Coyle

[1]This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero–the Irish Software Engineering Research Centre (www.lero.ie).

http://www.springer.com/978-1-4471-2296-8

Conquering Complexity
Hinchey, M.; Coyle, L. (Eds.)
2012, XXIV, 468 p., Hardcover
ISBN: 978-1-4471-2296-8