Conquering Complexity

Mike Hinchey · Lorcan Coyle
Editors

Conquering Complexity

Foreword by Roger Penrose

Springer

Editors
Mike Hinchey
Lero, Irish Software Eng Research Centre
University of Limerick
Limerick, Ireland
mike.hinchey@lero.ie

Lorcan Coyle
Lero, International Science Centre
University of Limerick
Limerick, Ireland
lorcan.coyle@lero.ie

ISBN 978-1-4471-2296-8 e-ISBN 978-1-4471-2297-5
DOI 10.1007/978-1-4471-2297-5
Springer London Dordrecht Heidelberg New York

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

Library of Congress Control Number: 2011944434

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Today, "complexity" is a word that is much in fashion. We have learned very well that many of the systems that we are trying to deal with in our contemporary science and engineering are very complex indeed. They are so complex that it is not obvious that the powerful tricks and procedures that served us for four centuries or more in the development of modern science and engineering will enable us to understand and deal with them...

... We are learning that we need a science of complex systems and we are beginning to develop it.

– Herbert A. Simon

Foreword

The year 2012—of publication of this book *Conquering Complexity*—is particularly distinguished by being the centenary year of Alan Turing, whose theoretical analysis of the notion of "computing machine", together with his wartime work in deciphering German codes, has had a huge impact on the enormous development of electronic computers, and the consequent impact that these devices have had on our lives, particularly with regard to science and technology. It is now possible to model extremely complex systems, whether they be naturally occurring physical processes or the predicted behaviour of human-constructed machinery. The complexity that can now be handled by today's electronic computers has completely transformed our understanding of many different kinds of physical behaviour, such behaviour being taken to act in accordance with the known physical laws. The extreme precision of these laws, as ascertained in numerous delicate experiments, allows us to have very considerable confidence in the results of these computations, and when the computations are done correctly, we may have a justified trust in the expectation of agreement between the computationally predicted outcomes and the details of observed behaviour. Conversely, such agreement between calculated predictions and actual physical behaviour reflects back as further confirmation on the very accuracy of the laws that are employed in the calculations.

However, the very possibility of reliably performing calculations of the extreme complication that is frequently required raises numerous new issues. Many of these issues would not have been evident before the advent of modern electronic computer technology, which has rendered it possible—and indeed commonplace—to enact the vast computations that are frequently needed. Whereas, our modern computers can be trusted to perform the needed calculations with enormous speed and accuracy, the machines themselves have no understanding of what they are doing nor of the purposes to which the results of these computations are to be put. It is *we* who must supply this understanding. Our particular choices of the actual computations that are to be performed need to be correct ones that do actually reflect the physical processes that are intended to be simulated. In addition, there are frequently many different ways of achieving the same ends, and insight and subtle judgements need to be employed in the decisions as to which procedures are the most effective to be

deployed. In my own extremely limited experience, in early 1956, when computer technology was still in its infancy, I obtained some direct experience of the vast simplification, even then, that could sometimes be achieved by the reformulation of a particular calculation into a subtly different one. How much greater is the potential, now, to improve the speed, accuracy—and indeed the very feasibility—of an intended simulation. The very enormity of the complexity of so many currently required computations vastly increases the role of such general considerations, these often leading to reliable computations that might have otherwise appeared not to be feasible, and frequently providing a much better understanding of what can indeed be achieved in practise. Many such matters are considered in this book, which address the issue of computational complexity from a great many different points of view. It is fascinating to see the variety of different types of argument that are here brought to bear on the issues involved, which so frequently indeed provide the taming of complexity in its multifarious forms.

Roger Penrose

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).

Contents

Contributors

John A. Anderson Cougaar Software, Inc., Falls Church, VA, USA, janderson@cougaarsoftware.com

Steve Boßelmann TU Dortmund, Dortmund, Germany, steve.bosselmann@cs.tu-dortmund.de

Shawn Bohner Rose-Hulman Institute of Technology, Terre Haute, USA, bohner@rose-hulman.edu

Jonathan P. Bowen Museophile Limited, London, UK, jonathan.bowen@lsbu.ac.uk

Karin K. Breitman Department of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil, karin@inf.puc-rio.br

Manfred Broy Institut für Informatik, Technische Universität München, München, Germany, broy@in.tum.de

Alan Burns Department of Computer Science, University of York, York, UK, burns@cs.york.ac.uk

Todd Carrico Cougaar Software, Inc., Falls Church, VA, USA, tcarrico@cougaarsoftware.com

Marco A. Casanova Department of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil, casanova@inf.puc-rio.br

Markus Doedt TU Dortmund, Dortmund, Germany, markus.doedt@cs.tu-dortmund.de

José Luiz Fiadeiro Department of Computer Science, University of Leicester, Leicester, UK, jose@mcs.le.ac.uk

Antonio Filieri DeepSE Group @ DEI, Politecnico di Milano, Milan, Italy, filieri@elet.polimi.it

Barry D. Floyd Orfalea College of Business, California Polytechnic University, San Luis Obispo, CA, USA, bfloyd@calpoly.edu

Antonio L. Furtado Department of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil, furtado@inf.puc-rio.br

Carlo Ghezzi DeepSE Group @ DEI, Politecnico di Milano, Milan, Italy, ghezzi@elet.polimi.it

Mike Hinchey Lero—the Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland, mike.hinchey@lero.ie

Gerard J. Holzmann Laboratory for Reliable Software, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA, gholzmann@acm.org

Michael Jackson The Open University, Milton Keynes, UK, jacksonma@acm.org

Mark Lawford McMaster University, Hamilton, ON, Canada, lawford@mcmaster.ca

Renato Levy Intelligent Automation Inc., Rockville, USA, rlevy@i-a-i.com

José A. F. de Macêdo Department of Computing, Federal University of Ceará, Fortaleza, CE, Brazil, jose.macedo@lia.ufc.br

Tom Maibaum McMaster University, Hamilton, ON, Canada, tom@maibaum.org

Tiziana Margaria Chair Service and Software Engineering, University of Potsdam, Potsdam, Germany, margaria@cs.uni-potsdam.de

Andrew Milluzzi Rose-Hulman Institute of Technology, Terre Haute, USA, milluzaj@rose-hulman.edu

Raffaela Mirandola DeepSE Group @ DEI, Politecnico di Milano, Milan, Italy, mirandola@elet.polimi.it

Joaquin Peña University of Seville, Seville, Spain, joaquinp@us.es

James L. Rash NASA Goddard Space Flight Center, Emeritus Greenbelt, MD, USA, james.l.rash@nasa.gov

Ramya Ravichandar CISCO Inc., San Jose, CA, USA, ramyar@vt.edu

Christopher A. Rouff Lockheed Martin Advanced Technology Laboratories, Arlington, VA, USA, christopher.rouff@lmco.com

Antonio Ruiz-Cortés University of Seville, Seville, Spain, aruiz@us.es

Norman F. Schneidewind Department of Information Science, Graduate School of Operational and Information Sciences, Monterey, CA, USA, ieeelife@yahoo.com

Bernhard Steffen Chair Programming Systems, TU Dortmund, Dortmund, Germany, steffen@cs.tu-dortmund.de

Roy Sterritt School of Computing and Mathematics, University of Ulster, Newtownabbey, Northern Ireland, r.sterritt@ulster.ac.uk

Giordano Tamburrelli DeepSE Group @ DEI, Politecnico di Milano, Milan, Italy, tamburrelli@elet.polimi.it

Walter F. Truszkowski NASA Goddard Space Flight Center, Emeritus Greenbelt, MD, USA, walter.f.truszkowski@nasa.gov

Vânia M.P. Vidal Department of Computing, Federal University of Ceará, Fortaleza, CE, Brazil, vvidal@lia.ufc.br

Alan Wassyng McMaster University, Hamilton, ON, Canada, wassyng@mcmaster.ca

Kun Wei Department of Computer Science, University of York, York, UK, kun@cs.york.ac.uk

Jim Woodcock Department of Computer Science, University of York, York, UK, jim@cs.york.ac.uk

Abbreviations

ABAP	Advanced Business Application Programming
ACM	Association for Computing Machinery
ADL	Architecture Description Language
ADT	Abstract Data Type
AE	Autonomic Element
ANS	Autonomic Nervous System
ANTS	Autonomous Nano-Technology Swarm
AOP	Aspect Oriented Programming
AOSE	Agent-Oriented Software Engineering
APEX	Adaptive Planning and Execution
API	Application Programming Interface
AUML	Agent UML
BAPI	Business Application Programming Interface
BB	Black-Box
BOR	Business Object Repository
BP	Business Process
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMS	Business Process Management System
CACM	Communications of the ACM
CAS	Complex Adaptive System
CASE	Computer-Aided Software Engineering
CBD	Component-Based Development
CCF	Common Cause Failure
CCFDB	Common-Cause Failure Data Base
CE	Capabilities Engineering
CMDA	Cougaar Model-Driven Architecture
COM	Computation Independent Model
COP	Common Operating Picture
CORBA	Common Object Request Broker Architecture
COTS	Component Off The Shelf

CPR	Core Plan Representation
CSP	Communicating Sequential Processes
CTMCs	Continuous Time Markov Chains
DARPA	Defense Advanced Research Projects Agency
DoD	Department of Defense
DL	Description Logic
DSL	Domain Specific Language
DST	Decision Support Tool
DTMCs	Discrete Time Markov Chains
EDAM	EMBRACE Ontology for Data and Methods
EMBRACE	European Model for Bioinformatics Research and Community Education
EMBOSS	European Molecular Biology Open Software Suite
EMF	Encore Modelling Language
ER	Entity-Relationship
ERP	Enterprise Resource Planning
FAST	Formal Approaches to Swarm Technologies
FD	Function Decomposition
FLG	Feature Level Graph
FDR	Failures-Divergences Refinement
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
GB	Grey-Box
GCAM	General Cougaar Application Model
GCME	Graphical Cougaar Model Editor
GDAM	General Domain Application Model
GEF	Graphical Editing Framework
GPAC	General-Purpose Autonomic Computing
GRASP	General Responsibility Assignment Software Patterns
GUI	Graphical User Interface
HITL	Human In The Loop
HOL	Higher Order Logic
HPRC	High-Performance Reconfigurable Computing
HRSM	Hubble Robotic Servicing Mission
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IT	Information Technology
IWIM	Idealised Worked Idealised Manager
jABC	Java Application Building Centre
JC3IEDM	Joint Consultation, Command and Control Information Exchange Data Model
JDBC	Java Database Connectivity
JDL	Joint Directors of Laboratories
JET	Java Emitter

jETI	Java Electronic Tool Integration Platform
JVM	Java Virtual Machine
JMS	Java Message Service
KLOC	Thousand (k) Lines of Code
LARA	Lunar Base Activities
LOC	Lines of Code
LOGOS	Lights-Out Ground Operating System
MAPE	Monitor-Analyse-Plan-Execute
MAS	Multi-Agent System
MBE	Model-Based Engineering
MBEF-HPRC	Model-Based Engineering Framework for High-Performance Reconfigurable Computing
MBSE	Model-Based Software Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDPs	Markov Decision Processes
MDSD	Model-Driven Software Development
MGS	Mars Global Surveyor
MIL	Module Interconnection Language
MIP	Multilateral Interoperablity Programme
MLM	Military Logistics Model
MPS	Meta Programming System
MOF	Meta Object Facility
MTBF	Mean-Time Between Failure
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organisation
NOS	Network Object Space
OASIS	Organisation for the Advancement of Structured Information Standards
OCL	Object Constraint Language
OMG	Object Management Group
OO	Object-Oriented
OOP	Object-Oriented Programming
OOram	Object Oriented Role Analysis and Modelling
OSMA	NASA Office of Systems and Mission Assurance
OTA	One-Thing Approach
OWL	Web Ontology Language
PAM	Prospecting Asteroid Mission
PARSY	Performance Aware Reconfiguration of software SYstems
PCTL	Probabilistic Computation Tree Logic
PDA	Personal Digital Assistant
PIM	Platform Independent Model
PLD	Programmable Logic Device
PSM	Platform Specific Model
PTCTL	Probabilistic Timed Computation Tree Logic

PVS	Prototype Verification System
QNs	Queueing Networks
QoS	Quality of Service
QSAR	Quantitative Structure Activity Relationships
R2D2C	Requirements-to-Design-to-Code
RC	Reconfigurable Computing
RFC	Remote Function Call
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSL	RAISE Specification Language
SASSY	Self-Architecting Software SYstems
SBS	Service-Based Systems
SC	Situation Construct
SCA	Service Component Architecture
SCADA	Supervisory Control and Data Acquisition
SDE	Shared Data Environment
SDR	Software-Defined Radio
SIB	Service-Independent Building block
SLA	Service Level Agreement
SLG	Service Level Graph
SNA	Social Networking Application
SNS	Semantic Network Space
SOAP	Simple Object Access Protocol
SOA	Service-Oriented Architecture
SOC	Service-Oriented Computing
SOS	Situational Object Space
SRF	Situational Reasoning Framework
SRML	SENSORIA Reference Modelling Language
SSA	Shared Situational Awareness
SWS	Semantic Web Service
TA	TeleAssistence
TCO	Total Cost of Ownership
TCTL	Timed Computation Tree Logic
$TCSP_M$	Timed CSP with the Miracle
UID	Unique Object Identifier
UML	Unified Modelling Language
URL	Uniform Resource Locator
UTP	Unifying Theories of Programming
VDM	Vienna Development Method
VHDL	VHSIC hardware description language
VLSI	Very-Large-Scale Integration
W3C	World Wide Web Consortium
WB	White-Box
WBS	White-Box Shared
WSDL	Web Service Definition Language

xADL	Extensible Architecture Description Language
XMDD	Extreme Model-Driven Development
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XP	Extreme Programming
XPDL	XML Process Definition Language
3GL	Third Generation Languages