CHAPMAN & HALL/CRC INNOVATIONS IN
SOFTWARE ENGINEERING AND SOFTWARE DEVELOPMENT

Computer Games and Software Engineering

Edited by
Kendra M. L. Cooper
Walt Scacchi

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Computer Games and Software Engineering

Chapman & Hall/CRC Innovations in Software Engineering and Software Development

Series Editor
Richard LeBlanc
Chair, Department of Computer Science and Software Engineering, Seattle University

AIMS AND SCOPE

This series covers all aspects of software engineering and software development. Books in the series will be innovative reference books, research monographs, and textbooks at the undergraduate and graduate level. Coverage will include traditional subject matter, cutting-edge research, and current industry practice, such as agile software development methods and service-oriented architectures. We also welcome proposals for books that capture the latest results on the domains and conditions in which practices are most effective.

PUBLISHED TITLES

Computer Games and Software Engineering
Kendra M. L. Cooper and Walt Scacchi

Software Essentials: Design and Construction
Adair Dingle

Software Metrics: A Rigorous and Practical Approach, Third Edition
Norman Fenton and James Bieman

Software Test Attacks to Break Mobile and Embedded Devices
Jon Duncan Hagar

Software Designers in Action: A Human-Centric Look at Design Work
André van der Hoek and Marian Petre

Fundamentals of Dependable Computing for Software Engineers
John Knight

Introduction to Combinatorial Testing
D. Richard Kuhn, Raghu N. Kacker, and Yu Lei

Building Enterprise Systems with ODP: An Introduction to Open Distributed Processing
Peter F. Linington, Zoran Milosevic, Akira Tanaka, and Antonio Vallecillo

Software Engineering: The Current Practice
Václav Rajlich

Software Development: An Open Source Approach
Allen Tucker, Ralph Morelli, and Chamindra de Silva

CHAPMAN & HALL/CRC INNOVATIONS IN
SOFTWARE ENGINEERING AND SOFTWARE DEVELOPMENT

Computer Games and Software Engineering

Edited by

Kendra M. L. Cooper
University of Texas
Dallas, USA

Walt Scacchi
University of California, Irvine
Irvine, USA

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2015 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Version Date: 20150317

International Standard Book Number-13: 978-1-4822-2669-0 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (http://www.copyright.com/) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

Contributors

Jonathan Bell
Department of Computer Science
Columbia University
New York, New York

Judith Bishop
Microsoft Research
Redmond, Washington

Bernd Brügge
Computer Science Department
Technical University of Munich
Munich, Germany

Kendra M.L. Cooper
Department of Computer Science
University of Texas, Dallas
Richardson, Texas

Jonathan de Halleux
Microsoft Research
Redmond, Washington

Thomas Debeauvais
Department of Informatics
University of California, Irvine
Irvine, California

Christopher Dragert
School of Computer Science
McGill University
Montréal, Québec, Canada

Robert J. Hall
AT&T Labs Research
Florham Park, New Jersey

Damir Ismailović
Computer Science Department
Technical University of Munich
Munich, Germany

Gail Kaiser
Department of Computer Science
Columbia University
New York, New York

Jörg Kienzle
School of Computer Science
McGill University
Montréal, Québec, Canada

Shaun Longstreet
Center for Teaching and Learning
Marquette University
Milwaukee, Wisconsin

Cristina V. Lopes
Department of Informatics
University of California, Irvine
Irvine, California

Dennis Pagano
Computer Science Department
Technical University of Munich
Munich, Germany

Barbara Reichart
Computer Science Department
Technical University of Munich
Munich, Germany

Daniel M. Russell
Google, Inc.
Menlo Park, California

Walt Scacchi
Center for Computer Games and Virtual Worlds
University of California, Irvine
Irvine, California

Swapneel Sheth
Department of Computer Science
Columbia University
New York, New York

Nikolai Tillmann
Microsoft Research
Redmond, Washington

Arthur Valadares
Department of Informatics
University of California, Irvine
Irvine, California

Clark Verbrugge
School of Computer Science
McGill University
Montréal, Québec, Canada

Alf Inge Wang
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Bian Wu
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Tao Xie
University of Illinois at Urbana, Champaign
Champaign, Illinois

CHAPTER 1

Introducing Computer Games and Software Engineering

Kendra M.L. Cooper and Walt Scacchi

CONTENTS

1.1 EMERGING FIELD OF COMPUTER GAMES AND SOFTWARE ENGINEERING

Computer games (CGs) are rich, complex, and often large-scale software applications. CGs are a significant, interesting, and often compelling software application domain for innovative research in software engineering (SE) techniques and technologies. CGs are progressively changing

the everyday world in many positive ways (Reeves and Read 2009). Game developers, whether focusing on entertainment-market opportunities or game-based applications in nonentertainment domains such as education, health care, defense, or scientific research (serious games or games with a purpose), thus share a common community of interest in how to best engineer game software.

There are many different and distinct types of games, game engines, and game platforms, much like there are many different and distinct types of software applications, information systems, and computing systems used for business. Understanding how games as a software system are developed to operate on a particular game platform requires identifying what types of games (i.e., game genre) are available in the market. Popular game genres include action or first-person shooters, adventure, role-playing game (RPG), fighting, racing, simulations, sports, strategy and real-time strategy, music and rhythm, parlor (board and card games), puzzles, educational or training, and massively multiplayer online games (MMOGs). This suggests that knowledge about one type of game (e.g., RPGs such as *Dungeons and Dragons*) does not subsume, contain, or provide the game play experience, player control interface, game play scenarios, or player actions found in other types of games. Therefore, being highly skilled in the art of one type of game software development (e.g., building a turn-taking RPG) does not imply an equivalent level of skill in developing another type of game software (e.g., a continuous play twitch or action game). This is analogous to saying that if a software developer is skilled in payroll and accounting software application systems, this does not imply that such a developer is also competent or skilled in the development of enterprise database management or e-commerce product sales over the web systems. The differences can be profound, and the developers' skills and expertise narrowly specialized.

Conversely, similar games, such as card or board games, raise the obvious possibility for a single game engine to be developed and shared or reused to support multiple game kinds of a single type. Game engines provide a runtime environment and reusable components for common game-related tasks, which leaves the developers freer to focus on the unique aspects of their game. For example, the games checkers and chess are played on an 8×8 checkerboard; though the shape and appearance of the game play pieces differ and the rules of game play differ, the kinds of player actions involved in playing either chess or checkers are the same (picking a piece and moving it to a square allowed by the rules of the game). Therefore,

being skilled in the art of developing a game of checkers can suggest the ability or competent skill in developing a similar game like chess, especially if both games can use the same game engine. However, this is feasible only when the game engine is designed to allow for distinct sets of game rules and distinct appearance of game pieces—that is, the game engine must be designed for reuse or extension. This design goal is not always an obvious engineering choice, and it is one that increases the initial cost of game engine development (Bishop et al. 1998; Gregory 2009). Subsequently, developing software for different kinds of games of the same type, or using the same game engine, requires a higher level of technical skill and competence in software development than designing an individual game of a given type.

Understanding how game software operates on a game platform requires an understanding of the game device (e.g., Nintendo GameBoy, Microsoft Xbox One, Apple iPhone) and the internal software run-time environment that enables its intended operation and data communication capabilities. A game platform constrains the game design in terms of its architectural structure, how it functions, how the game player controls the game device through its interfaces (keyboard, buttons, stylus, etc.) and video/audio displays, and how they affect game data transmission and reception in a multiplayer game network.

1.2 BRIEF HISTORY OF COMPUTER GAME SOFTWARE DEVELOPMENT

Game software researchers and developers have been exploring computer game software engineering (CGSE) from a number of perspectives for many years. Many are rooted in the history of CG development, much of which is beyond what we address here, as are topics arising from many important and foundational studies of games as new media and as cultural practice. However, it may be reasonable to anticipate new game studies that focus on topics such as how best to develop CGs for play across global cultures or through multisite, global SE practices.

The history of techniques for CG software development goes back many decades, far enough to coincide with the emergence of SE as a field of research and practices in the late 1960s. As CG software development was new and unfamiliar, people benefitted from publications of open source game software, often written in programming languages such as Fortran (Spencer 1968). Before that, interest in computer-based playing against human opponents in popular parlor games such as chess,

checkers, poker, bridge, backgammon, and go was an early fascination of researchers exploring the potential of artificial intelligence (AI) using computers (Samuel 1960). It should be noted that these CG efforts did not rely on graphic interfaces, which were to follow with the emergence of video games that operated on general-purpose computer workstations, and later personal computers and special-purpose game consoles.

Spacewar!, PONG, Maze War, DOOM, SimCity, and thousands of other CGs began to capture the imagination of software developers and end users as opening up new worlds of interactive play for human player versus computer or player versus player game play to large public audiences, and later to end-user development or modification of commercial games (Burnett 2004).

Combat-oriented maze games such as *Maze War, Amaze* (Berglund and Cheriton 1985), *MiMaze* (Gautier and Dior 1998), and others (Sweeney 1998) helped introduce the development and deployment of networked multiplayer games. *BattleZone, Habitat,* and other game-based virtual worlds similarly helped launch popular interest in MMOGs (Bartle 1990), along with early social media capabilities such as online forums (threaded e-mail lists), multiuser chat (including Internet Relay Chat) and online chat meeting rooms (from multiuser dungeons), which would then be globally popularized within *Ultima Online, EverQuest, World of Warcraft,* and others. The emergence of the CG development industry, with major studios creating games for global markets, soon made clear the need for game development to embrace modern SE techniques and practices, or else likely suffer the fate of problematic, difficult-to-maintain or -expand game software systems, which is the common fate of software application systems whose unrecognized complexity grows beyond the conventional programming skills of their developers.

As many game developers in the early days were self-taught software makers, it was not surprising to see their embrace of practices for sharing game source code and play mechanic algorithms. Such ways and means served to collectively advance and disseminate game development practices on a global basis. As noted above, early game development books prominently featured open source game programs that others could copy, build, modify, debug, and redistribute, albeit through pre-Internet file sharing services such as those offered by CompuServe, though game-making students in academic settings might also share source code to games such as *Spacewar!, Adventure,* and *Zork* using Internet-accessible file servers via file transfer protocols.

The pioneering development of *DOOM* in the early 1990s (Hall 1992; Kushner 2003), along with the growing popularity of Internet-based file sharing, alongside of the emergence of the open source software movement, the World Wide Web, and web-based service portals and applications, all contributed in different ways to the growing realization that CGs as a software application could similarly exploit these new ways and means for developing and deploying game software systems. Id Software, through the game software developer John Carmack and the game designer John Romero, eventually came to realize that digital game distribution via file sharing (initially via floppy disks for freeware and paid versions of *DOOM*), rather than in-store retail sales, would also point the way to offload the ongoing development and customization of games such as *DOOM*, by offering basic means for end-user programming and modification of CGs that might have little viable commercial market sales remaining (Au 2002; Kushner 2003). The end users' ability to therefore engage in primitive CGSE via game modding was thus set into motion (cf. Au 2002; Burnett 2004; Morris 2003; Scacchi 2010). Other game development studios such as Epic Games also began to share their game software development tools as software development kits (SDKs), such as the UnrealEd game level editor and script development interface, and its counterpart packages with *Quake* from Id Software (QuakeEd) and *Half-Life* from Valve Software. These basic game SDKs were distributed for no additional cost on the CD-ROM media that retail consumers would purchase starting in the late 1990s. Similarly, online sharing of game software, as either retail product or free game mod, was formalized by Valve Software through their provision of the Steam online game distribution service, along with its integrated payment services (Au 2002; Scacchi 2010).

Finally, much of the wisdom to arise from the early and more recent days of CG development still focus attention on game programming and game design, rather than on CGSE. For example, the current eight-volume series *Game Programming Gems*, published by Charles River Media and later Cengage Learning PTR (2000–2010), reveals long-standing interest on the part of game makers to view their undertaking as one primarily focused on programming rather than SE; field of SE long ago recognized that programming is but one of the major activities in developing, deploying, and sustaining large-scale software system applications, but not the only activity that can yield high quality software products and related artifacts. Similarly, there are many books written by well-informed, accomplished game developers on how best to design games as *playful interactive media* that can

induce fun or hedonic experiences (Fullerton et al. 2004; Meigs 2003; Rogers 2010; Salen and Zimmerman 2004; Schell 2008). This points to another gap, as many students interested in making CG choose to focus their attention toward a playful user experience, while ignoring whether SE can help produce better quality CG at lower costs with greater productivity. That is part of the challenge that motivates new research and practice in CGSE.

1.3 TOPICS IN COMPUTER GAMES AND SOFTWARE ENGINEERING

This book collects 11 chapters that systematically explore the CGSE space. The chapters that follow draw attention to topics such as CG and SE education (SEE), game software requirements engineering, game software architecture and design approaches, game software testing and usability assessment, game development frameworks and reusability techniques, and game scalability infrastructure, including support for mobile devices and web-based services. Here, a sample of earlier research efforts in CGSE that help inform these contemporary studies is presented in the following subsections.

1.3.1 Computer Games and SEE

Swartout and van Lent (2003) were among the earliest to recognize the potential of bringing CG and game-based virtual worlds into mainstream computer science education and system development expertise. Zyda (2006) followed by further bringing attention to the challenge of how best to educate a new generation of CG developers. He observes something of a conflict between programs that stress CG as interactive media created by artists and storytellers (therefore somewhat analogous to feature film production) and programs that would stress the expertise in computer science required of game software developers or infrastructural systems engineers. These pioneers in computer science research recognized the practical utility of CG beyond entertainment that could be marshaled and directed to support serious game development for training and educational applications. However, for both of these visions for undergraduate computer science education, SE has little role to play in their respective framings. In contrast, SE faculty who teach project-oriented SE courses increasingly have sought to better motivate and engage students through game software development projects, as most computer science students worldwide are literate in CG and game play. Building from this insight, Oh Navarro and van der Hoek (2005, 2009), the Claypools (Claypool and

Claypool 2005), and Wang and students (Wang 2011; Wang et al. 2008) were among the earliest to call out the opportunity for focusing on the incorporation of CG deep into SEE coursework.

Oh Navarro and van der Hoek started in the late 1990s exploring the innovative idea of teaching SE project dynamics through a simulation-based SE RPG, called *SimSE*. Such a game spans the worlds of software process modeling and simulation, team-based SE, and SE project management, so that students can play, study, and manipulate different SE tasking scenarios along with simulated encounters with common problems in SE projects (e.g., developers falling behind schedule, thus disrupting development plans and inter-role coordination). In this way, SE students could play the game before they undertook the software development project, and thus be better informed about some of the challenges of working together as a team, rather than just as skilled individual software engineers.

The Claypools highlight how SE project or capstone courses can focus on student teams conducting game development projects, which seek to demonstrate their skill in SE, as well as their frequent enthusiastic interest in CG culture and technology. The popularity of encouraging game development projects for SE capstone project courses is now widespread. However, the tension between CG design proffered in texts that mostly ignore modern SE principles and practices (Fullerton et al. 2004; Meigs 2003; Rogers 2010; Salen and Zimmerman 2004; Schell 2008) may sometimes lead to projects that produce interesting, playful games but do so with minimal demonstration of SE skill or expertise.

Wang et al. (2008) have demonstrated how other CG and game play experiences can be introduced into computer science or SEE coursework through gamifying course lectures that facilitate faculty–student interactions and feedback. Wang (2011) along with Cooper and Longstreet (2012) (and in Chapter 3) expand their visions for SEE by incorporating contemporary SE practices such as software architecture and model-driven development. More broadly, Chapters 2 through 6 all discuss different ways and means for advancing SEE through CG.

Finally, readers who teach SEE project courses would find it valuable to have their students learn CGSE through their exposure to the history of CG software development, including a review of some of the pioneering papers or reports cited earlier in this introductory chapter. Similarly, whether to structure the SEE coursework projects as massively open online courses or around competitive, inter-team game jams also merits consideration. Such competitions can serve as test beds for empirical

SE (or SEE) studies, for example, when project teams are composed by students who take on different development roles and each team engages members with comparable roles and prior experience. Such ideas are discussed in Chapter 12.

1.3.2 Game Software Requirements Engineering

Understanding how best to elicit and engineer the requirements for CG is unsurprisingly a fertile area for CGSE research and practice (Ampatzoglou and Stamelos 2010; Callele et al. 2005), much like it has been for mainstream SE. However, there are still relatively few game development approaches that employ SE requirements development methods such as use cases and scenario-based design (Walker 2003).

Many game developers in industry have reviewed the informal game "postmortems" that first began to be published in *Game Developer* magazine in the 1990s (Grossman 2003), and more recently on the Gamasutra.com online portal. Grossman's (2003) collection of nearly 50 postmortems best reveals common problems that recur in game development projects, which cluster around project software and content development scheduling, budget shifts (generally development budget cuts), and other nonfunctional requirements that drift or shift in importance during game development projects (Alspaugh and Scacchi 2013; Petrillo et al. 2009). None of this should be surprising to experienced SE practitioners or project managers, though it may be "new knowledge" to SE students and new self-taught game developers. Similarly, software functional requirements for CG most often come from the game producers or developers, rather than from end users. However, nonfunctional requirements (e.g., the game should be fun to play but hard to master and it should be compatible with mobile devices and the web) dominate CG development efforts, and thus marginalize the systematic engineering of functional game requirements. Nonetheless, the practice of openly publishing and sharing postproject descriptions and hindsight rationalizations may prove valuable as another kind of empirical SE data for further study, as well as something to teach and practice within SEE project courses.

1.3.3 Game Software Architecture Design

CGs as complex software applications often represent configurations of multiple software components, libraries, and network services. As such, CG software must have an architecture, and ideally such an architecture is explicitly represented and documented as such. Although such

architecture may be proprietary and thus protected by its developers as intellectual property covered by trade secrets and end-user license agreements, there is substantial educational value in having access to such architectural renderings as a means for quickly grasping key system design decisions and participating modules in game play event processing. This is one reason for interest in games that are open to modding (Seif El-Nasr and Smith 2006; Scacchi 2010). However, other software architecture concerns exist. For instance, there are at least four kinds of CG software architecture that arise in networked multiplayer games: (1) the static and dynamic run-time architectures for a game engine; (2) the architecture of the game development frameworks or SDKs that embed a game's development architecture together with its game engine (Wang 2011); (3) the architectural distribution of software functionality and data processing services for networked multiplayer games; and (4) the informational and geographical architecture of the game levels as designed play spaces. For example, for (3) there are four common alternative system configurations: single server for multiple interacting or turn-taking players, peer-to-peer networking, client–server networking for end-user clients and playspace data exchange servers, and distributed, replicated servers for segmented user community play sessions (via *sharding*) (Alexander 2003; Bartle 1990; Berglund and Cheriton 1985; Bishop et al. 1998; Gautier and Dior 1998; Hall 1992; Sweeney 1998).

In contrast, the focus on CG as interactive media often sees little or no software architecture as being relevant to game design, especially for games that assume a single server architecture or PC game run-time environment, or in a distributed environment that networking system specialists, it is assumed, will design and provide (Fullerton et al. 2004; Meigs 2003; Rogers 2010; Salen and Zimmerman 2004; Schell 2008). Ultimately, our point is not to focus on the gap between game design and game software (architecture) design as alternative views but to draw attention to the need for CGSE to find ways to span the gap.

1.3.4 Game Software Playtesting and User Experience

CGs as complex software applications for potentially millions of end users will consistently and routinely manifest bugs (Lewis 2010). Again, this is part of the puzzle of any complex SE effort, so games are no exception. However, as user experience and thus user satisfaction may be key to driving viral social media that helps promote retail game sales and adoption, paying close attention to bugs and features in CG development and

usability (Pinelle et al. 2008) may be key to the economic viability of a game development studio. Further, on the basis of decades of experience in developing large-scale software applications, we believe that most end users cannot articulate their needs or requirements in advance but can assess what is provided in terms of whether or not it meets their needs. This in turn may drive the development of large-scale, high-cost CGs that take calendars to produce and person-decades (or person-centuries) of developer effort away from monolithic product development life cycles to ones that are much more incremental and driven by user feedback based on progressively refined or enhanced game version (or prototype) releases. Early and ongoing game playtesting will likely come to be a central facet of CGSE, as will tools and techniques for collecting, analyzing, and visualizing game playtesting data (Drachen and Canossa 2009; Zoeller 2013). This is one activity where CGSE efforts going forward may substantially diverge from early CG software development approaches, much like agile methods often displace *waterfall* software life cycle development approaches. Therefore, CG developers, much like mainstream software engineers, are moving toward incremental development, rapid release, and user playtesting to drive new product release versions.

1.3.5 Game Software Reuse

Systematic software reuse could be considered within multiple SE activities (requirements, architecture, design, code, build and release, test cases) for a single game or a product line of games (Furtado et al. 2011). For example, many successful CGs are made into *franchise brands* through the production and release of extension packs (that provide new game content or play levels) or product line sequels (e.g., *Quake, Quake II, and Quake III; Unreal, Unreal Tournament 2003, and Unreal Tournament 2007*). Whether or how the concepts and methods of software product lines can be employed in widespread CG business models is unclear and underexplored. A new successful CG product may have been developed and released in ways that sought to minimize software production costs, thus avoiding the necessary investment to make the software architecture reusable and extensible and the component modules replaceable or upgradable without discarding much of the software developed up to that point. This means that SE approaches to CG product lines may be recognized in hindsight as missed opportunities, at least for a given game franchise.

Reuse has the potential to reduce CG development costs and improve quality and productivity, as it often does in mainstream SE. Commercial

CG development relies often on software components (e.g., game engines) or middleware products provided by third parties (AI libraries for nonplayer characters [NPCs]) as perhaps its most visible form of software reuse practice. Game SDKs, game engines, procedural game content generation tools, and game middleware services all undergo active R&D within industry and academia. Game engines are perhaps the best success story for CG software reuse, but it is often the case that commercial game development studios and independent game developers avoid adoption of such game engines when they are perceived to overly constrain game development patterns or choice of game play mechanics to those characteristic of the engine. This means that game players may recognize such games as offering derivative play experience rather than original play experience. However, moving to catalogs of pattern or antipatterns for game requirements, architecture and design patterns for game software product lines (Furtado et al. 2011), and online repositories of reusable game assets organized by standardized ontologies may be part of the future of reusable game development techniques. As noted earlier, such topics are explored in Chapters 10 and 11.

Other approaches to software reuse may be found in free or open source software for CG development (Scacchi 2004), and also in AI or computational intelligence methods for semiautomated or automated content generation and level design (IEEE 2014).

1.3.6 Game Services and Scalability Infrastructure

CGs range from small-scale, stand-alone applications for smart phones (e.g., app games) to large-scale, distributed, real-time MMOGs. CGs are sometimes played by millions of end users, so that large-scale, *big data* approaches to game play analytics and data visualization become essential techniques for engineering sustained game play and deployment support (Drachen and Canossa 2009; Zoeller 2013). Prior knowledge of the development of multiplayer game software systems and networking services (cf. Alexander 2003; Berglund and Cheriton 1985; Gautier and Dior 1998; Sweeney 1998) may be essential for CGSE students focusing on development of social or mobile MMOGs. In order to engage the users and promote the adoption and ongoing use of such large and upward or downward scalable applications, CGSE techniques have significant potential but require further articulation and refinement. Questions on the integration of game playtesting and end-user play analytic techniques together with large-scale, big-data applications are just beginning to emerge. Similarly, how best to design back-end game data management capabilities or

remote middleware game play services also points to SE challenges for networked software systems engineering, as has been recognized within the history of networked game software development (Alexander 2003, Bartle 1990, Berglund and Cheriton 1985; Gautier and Dior 1998; Sweeney 1998). Whether or how cloud services or cloud-based gaming has a role in CGSE may benefit by review of the chapters that follow.

The ongoing emphasis on CGs that realize playful, fun, social, or learning game experiences across different game play platforms leads naturally to interdisciplinary approaches to CGSE, where psychologists, sociologists, anthropologists, and economists could provide expertise on defining new game play requirements and experimental designs to assess the quality of user play experiences. Further, the emergence of online fantasy sports, along with eSports (e.g., team/player vs. team/player competitions for prizes or championship rankings) and commercial endeavors such as the National Gaming League for professional-level game play tournaments, points to other CGSE challenges such as cheat prevention, latency equalization, statistical scoring systems, complex data analytics (DsC09), and play data visualizations (Zoeller 2013), all of which support game systems that are balanced and performance (monitoring) equalized for professional-level tournaments. The social sciences could provide insight into how to attract, engage, and retain players across demographic groups (e.g., age, gender, geographic location), much like recent advances in the Cooperative and Human Aspects in Software Engineering workshop and ethnographic studies of users in contemporary SE research.

With this background in mind, we turn to explain the motivating events that gave rise to the production of this book on CGSE.

1.4 EMERGENCE OF A COMMUNITY OF INTEREST IN CGSE

At the core of CGs are complex human–software platform interactions leading to emergent game play behaviors. This complexity creates difficulties architecting game software components, predicting their behaviors, and testing the results. SE has not yet been able to meet the demands of the CG software development industry, an industry that works at the forefront of technology and creativity, where creating a fun experience is the most important metric of success. In recognition of this gap, the first games and software engineering workshop (GAS 2011) was held at the International Conference on Software Engineering (ICSE 2011), initiated through the efforts of Chris Lewis and E. James Whitehead (both from UC Santa Cruz). Together with a committee of like-minded others

within the SE community, Lewis and Whitehead sought to bring together SE researchers interested in exploring the demands of game creation and ascertain how the SE community can contribute to this important creative domain. GAS 2011 participants were also challenged to investigate how games can help aid the SE process or improve SEE. Research in these areas has been exciting and interesting, and GAS 2011 was envisioned to be the first time practitioners from these fields would have the opportunity to come together at ICSE to investigate the possibilities of this innovative research area. The content of Chapters 4 and 8 was originally presented at GAS 2011, in simpler form.

The GAS 2012 workshop explored issues that crosscut the SE and the game engineering communities. Advances in game engineering techniques can be adopted by the SE community to develop more engaging applications across diverse domains: education, health care, fitness, sustainable activities (e.g., recycling awareness), and so on. Successful CGs feature properties that are not always found in traditional software: they are highly engaging, they are playful, and they can be fun to play for extended periods of time. Engaging games enthrall players and result in users willing to spend increasing amounts of time and money playing them. ICSE 2012 sought to provide a forum for advances in SE for developing more sustainable (*greener*) software, so GAS 2012 encouraged presentation and discussion of ways and means through green game applications. For example, approaches that support adapting software to trade off power consumption and video quality would benefit the game community. SE techniques spanning patterns (requirements, design), middleware, testing techniques, development environments, and processes for building sustainable software are of great interest. Chapters 6 and 10 were both initially presented in simpler form at GAS 2012.

GAS 2013 explored issues that crosscut the SE and the game development communities. Advances in game development techniques can be adopted by the SE community to develop more engaging applications across diverse domains: education, health care, fitness, sustainable activities (e.g., recycling awareness), and so on. GAS 2013 provided a forum for advances in SE for developing games that enable progressive societal change through fun, playful game software. SE techniques spanning patterns, middleware, testing techniques, development environments, and processes were in focus and consumed much of participant interest, including a handful of live game demonstrations. Chapters 9 and 5 were initially presented in simpler form at GAS 2013. Chapters 2, 7,

and Chapter 11 are new and were prepared specifically for this book. Finally, it should be noted that Cooper, Scacchi, and Wang were the co-organizers of GAS 2013.

The topic of how best to elevate the emerging results and discipline of CGSE was put into motion at the end of GAS 2013; this book is now the product of that effort. Many participants at the various GAS workshops were invited to develop and refine their earlier contributions into full chapters. The chapters that follow are the result. Similarly, other research papers that speak to CGSE topics that appeared in other workshops, conferences, or journals were reviewed for possible inclusion in this book. Therefore, please recognize the chapters that follow as a sample of recent research in the area of CGSE, rather than representing some other criteria for selection. However, given more time and more pages to fill for publication, others who were not in a position to prepare a full chapter of their work would have been included.

As such, we turn next to briefly introduce each of the chapters that were contributed for this book on CGSE. The interested reader is encouraged to consider focusing on topics of greatest interest first and to review the other chapters as complementary issues found at the intersection of CG and SE covered across the set of remaining chapters.

1.5 INTRODUCING THE CHAPTERS AND RESEARCH CONTRIBUTIONS

A comprehensive literature review of CG in software education is presented in Chapter 2 by Alf Inge Wang and Bian Wu. They explore how CG development is being integrated into computer science and SE coursework. The survey is organized around three research questions:

- The first question focuses on discovering the topics where game development has been used as a teaching method. These results are presented in three categories: computer science (37 articles), SE (16 articles), and applied computer science (13 articles). For computer science, a variety of topics (e.g., programming, AI, algorithms) are being taught at different levels (university and elementary, middle, and high school). Game development approaches in university courses on programming dominate the findings, followed by AI. For SE, a variety of topics (e.g., architecture, object-oriented analysis and design, and testing) are being taught in university courses. Game development approaches in design topics (architecture and

object-oriented) lead the findings, followed by testing. For applied computer science a variety of topics (e.g., game design, game development with a focus on game design, and art design) are being taught in pre-college/university and university courses. These approaches focus on creating or changing games through graphical tools to create terrains, characters, game objects, and populate levels. Applied courses on game design and development dominate the findings, followed by art design; approximately half the findings were for courses at the pre-college/university level.

- The second research question focuses on identifying the most common tools used and any shared experiences from using these tools. The articles reveal a plethora of game development frameworks and languages in use. Interestingly, the most commonly used frameworks include the educators' own framework, XNA, or a Java game development framework; Unity has not been reported in the articles reviewed. With respect to programming languages, visual programming languages and Java dominate, followed by C#. Visual languages have worked well for introducing programming concepts, promoting the field of computer science. Often, students are asked to create simple 2D games from scratch; an alternative approach reported is to use game modding, in which the existing code is changed, modifying the behavior and presentation of a game.
- The third research question focuses on identifying common experiences from using game development to teach computer science and SE subjects. Most studies in the survey report that game development improves student motivation and engagement, as the visualization makes programming fun. However, only a few studies report learning improvements in terms of better grades; there is a tendency for some students to focus too much on game development instead of the topic being taught. In addition, many articles reported that game development positively supported recruiting and enrolment efforts in computer science and SE.

Based on the results of this survey, the authors propose a set of recommendations for choosing an appropriate game development framework to use in a course. The recommendations include the consideration of the educational goals, subject constraints, programming experience, staff expertise, usability of the game development platform, and the technical environment.

A model-driven SE approach to the development of serious educational games (SEGs) is presented in Chapter 3 by Kendra Cooper and Shaun Longstreet. SEGs are complex applications; developing new ones has been time consuming and expensive, and has required substantial expertise from diverse stakeholders: game developers, software developers, educators, and players. To improve the development of SEGs, the authors present a model-driven engineering (MDE)-based approach that uniquely integrates elements of traditional game design, pedagogical content, and SE. In the SE community, MDE is an established approach for systematically developing complex applications, where models of the application are created, analyzed (validated/verified), and subsequently transformed to lower levels of abstraction.

The MDE-based approach consists of three main steps to systematically develop the SEGs:

- The first step is to create an informal model of the SEG captured as a storyboard with preliminary descriptions of the learning objectives, game play, and user interface concepts. The learning objectives cover specific topics (e.g., design patterns, grade 4 reading) as well as transferable skills (e.g., problem solving, analysis, critical thinking). Storyboards are an established, informal approach used in diverse creative endeavors to capture the flow of events over time using a combination of graphics and text. The SimSYS storyboard is tailored to explicitly include the learning objectives for the game.
- The second step is to transform the informal model into a semiformal, tailored unified modeling language (UML) use case model (visual and tabular, template-based specifications). Here, the preliminary description is refined to organize it into acts, scenes, screens, and challenges; each of these has a tabular template to assist in the game development. The templates include places for the learning objectives; they can be traced from the highest level (game template) down to specific challenges. More detailed descriptions of the game play narrative, graphics, animation, music and sound effects, and challenge content are defined.
- The third step is to transform the semiformal model into formal, executable models in statecharts and extensible markup language (XML). A statechart can undergo comprehensive simulation or animation to verify the model's behavior using existing tool support; errors can be identified and corrected in both the statechart model

and the semiformal model as needed. XML is the game specification, which can be loaded, played, and tested using the SimSYS game play engine; the XML schema definition for the game is defined.

A key feature of the MDE approach is the meta-model foundation, which explicitly represents traditional game elements (e.g., narrative, characters), educational elements (e.g., learning objectives, learning taxonomy), and their relationships. The approach supports the wide adoption across curricula, as domain-specific knowledge can be plugged in across multiple disciplines (e.g., science, technology, engineering and mathematics [STEM], humanities) and the thorough integration of learning objectives. This approach is flexible, as it can be applied in an agile, iterative development process by describing a part of the game informally, semiformally, and formally (executable), allowing earlier assessment and feedback on a running (partial) game.

In Chapter 4, Swapneel Sheth, Jonathan Bell, and Gail Kaiser present an experience report describing their efforts in using game play motifs, inspired from online RPGs, and competitive game tournaments to introduce students to software testing and design principles. The authors draw upon the reported success of gamifying another topic in SE (formal verification) by proposing a social approach to introduce students to software testing using their game-like environment HALO (highly addictive, socially optimized) SE. HALO can make the software development process, and in particular, the testing process, more fun and social by using themes from popular CGs. HALO represents SE tasks as quests; a storyline binds multiple quests together. Quests can be individual, requiring a developer to work alone, or in groups, requiring a developer to form a team and work collaboratively toward an objective. Social rewards in HALO can include titles—prefixes or suffixes of players' names—and levels, both of which showcase players' successes in the game world. These social rewards harness a model, operant conditioning, which rewards players for good behavior and encourages repeating good behavior.

HALO was introduced into the course as an optional part of two assignments and as a bonus question in a third assignment. The student evaluations on using HALO in their assignments revealed that the approach may be more effective if the HALO quests had a stronger alignment with all the students doing well in the assignment, not as an optional or bonus question that may only appeal to some of the students. The ability to embrace a broader range of students, perhaps by providing some adaptability to adjust the level of difficulty based on what the students would find it most

useful, was recommended by the authors. For example, students who are struggling with the assignment might want quests covering more basic aspects of the assignment, whereas students who are doing well might need quests covering more challenging aspects.

To instill good software design principles, a programming assignment using a game was used in combination with a competitive game tournament in an early course. The assignment and tournament centered on developing the game *Battleship*. The students were provided with three interfaces as a starting point for the assignment: game, location, and player. As long as the students' code respected the interfaces, they would be able to take part in the tournament. The teaching staff provided implementations of the game and location interfaces; each student's automated computer player implementation was used. Extra credit was used as an incentive; even though the extra credit was modest, the combination of the extra credit and the competitive aspect resulted in almost the entire class participating in the tournament: a remarkable 92% of the class had implementations that realized the defined interfaces and were permitted to compete in the tournament. The authors note that the competitive tournaments require substantial resources (e.g., time, automated testing frameworks, equipment), in particular for large classes.

In Chapter 5, Tao Xie, Nikolai Tillmann, Jonathan de Halleux, and Judith Bishop focus on the gamification of online programming exercise systems through their online CG, Pex4Fun, and its successor, Code Hunt. These game-based environments are designed to address educational tasks of teaching and learning programming and SE skills. They are open, browser based, interactive gaming-based teaching and learning platforms for .NET programming languages such as C#, Visual Basic, and F#. Students play coding-duel game play sessions, where they need to write code to implement the capabilities of a hidden specification (i.e., sample solution code not visible to the student). The Pex4Fun system automatically finds discrepancies in the behavior between the student's code and the hidden specification, which are provided as feedback to the student. The students then proceed to correct their code. The coding-duel game type within Pex4Fun is flexible and can be used to create games that target a wide range of skills such as programming, program understanding, induction, debugging, problem solving, testing, and specification writing, with different degrees of difficulty. Code Hunt offers additional gaming aspects to enhance the game play experience such as audio support, a leaderboard, and visibility to the coding duels of other players to enhance the

social aspect; games can also be organized in a series of worlds, sectors, and levels, which become increasingly challenging. Pex4Fun has been adopted as a major platform for assignments in a graduate SE course, and a coding-duel contest has recently been held at the ICSE 2011 for engaging conference attendees to solve coding duels in a dynamic social contest. The response from the broader community using the Pex4Fun system has been positive and enthusiastic, indicating the gamification of online programming exercise systems holds great promise as a tool in SEE.

An exploratory study on how human tutors interact with learners playing serious games is presented by Barbara Reichart, Damir Ismailović, Dennis Pagano, and Bernd Brügge in Chapter 6. In traditional educational settings, a professional human tutor observes a student's skills and uses those observations to select learning content, adapting the material as needed. Moving into a serious game educational setting, this study investigates how players can be characterized and how to provide them with help in this new environment. The study uses four small serious games with focus on elementary-school mathematics. The authors created these over a span of 2 years; the new games were needed to retain very high control over the game elements (content, difficulty level, and game speed), which would not be possible with games already available. Interviews with experts and observing children at play provided qualitative data for the first part of the study. Here, the results reveal that the human tutor observes the correct and incorrect execution of the tasks in the game as well as the motorical execution (hand–eye coordination, timing); tutors rate the skills of the learners in a fuzzy way. In the second part of the study, interviews with experts provided qualitative data; here, experts observed the recordings of children playing. The experts defined different levels of difficulty that they considered reasonable for each game. To provide the different levels of difficulty, a detailed description of the data (content) that can be changed in each of the developed serious games was defined. In addition to changes in the content, changes to some properties of the game elements are identified to affect specific skills. For example, adapting the speed of a game element has a direct effect on some skills necessary for mathematics, such as counting. Therefore, adapting the game element properties to change the level of difficulty is an option—a change in the learning content is not always necessary. Using the results of these studies, the authors also propose a definition for the adaptivity process in a serious game consisting of four stages: monitoring players (A1), learner characterization (A2), assessment generation (B1), and adaptive intervention (B2). This thorough, extensive

study provides a strong foundation for the community to build upon in the investigation of adapting serious games, with respect to research methodologies and the results reported.

A scalable architecture for MMOGs is presented by Thomas Debeauvais, Arthur Valadares, and Cristina V. Lopes in Chapter 7. The research considers how to harmonize the representational state transfer principles, which have been used very successfully for scaling web applications, with the architecture-level design of MMOGs. The proposed architecture, restful client–server architecture (RCAT), consists of four tiers: proxies, game servers, caches, and database. Proxies handle the communication with clients, game servers handle the computation of the game logic, and the database ensures data persistence. RCAT supports the scalability of MMOGs through the addition of servers that provide the same functionality. The authors developed a reference implementation of RCAT as a middleware solution, and then conducted experiments to characterize the performance and identify bottlenecks.

Two quantitative performance studies are reported. The first uses a simple MMOG to analyze the impact of the number of clients on the bandwidth: from the proxy to the clients and from the server to the database, with and without caching. The experiments show that the bandwidth from the proxy to the clients increases quadratically; the database can be a central bottleneck, although caching can be an effective strategy for this component. The second experiment evaluates the performance impact of scaling up the number of players using an RCAT reference application, which is a multiplayer online jigsaw puzzle game. These experiments are designed to quantify how the number of proxies and the number of game servers scale with the number of players (bots) for different message frequencies. The quantitative results summarize (1) the behavior of CPU utilization and round trip time (latency) as the number of clients increases, (2) CPU utilization and context switches per second as the number of clients increases, (3) the maximum number of clients supported under alternative core/machine scenarios and message frequencies, and (4) CPU utilization when the maximum capacity is reached.

The authors' proposal of the RCAT architecture, the development of a reference implementation, the development of a reference application, and a quantitative performance study provides the community with a scalable architectural solution with a rigorous validation. The game-agnostic approach to the RCAT architecture, modularizing the game logic in one tier, means that it can be applied broadly, supporting the development of diverse MMOGs.

The challenges in developing multiplayer outdoor smart phone games are presented in five core areas of SE (requirements, architecture, design, implementation, and testing) by Robert Hall in Chapter 8. The games, part of the Geocast Games Project, incorporate vigorous physical activity outdoors and encourage multiplayer interactions, contributing to worthwhile social goals. Given the outdoor environment these games are played in, such as parks or beaches, the games need to be deployed solely on equipment people are likely to carry anyway for other purposes, namely, smart phones and iPods, and they need to rely on device-to-device communication, as network access may not be available. These two characteristics have profound impacts on SE activities. One challenge in the requirements engineering area is the definition of domain-specific set of meta-requirements applicable to outdoor game design, providing domain-specific guidelines and constraints on requirements models to help developers better understand when they are posing impossible or impractical requirements for new games. The architectural challenges include defining solutions that support full distribution (no central server) and long-range play (seamless integration with networks when they are available). The design challenges include the need to allow coherent game behavior to be implemented at the top of a fully distributed architecture, subject to sporadic device communication. A collection of design issues falls under this distributed joint state problem: when devices have been out of communication, they need to resynchronize in a rapid and fair way when they reestablish a connection. The implementation challenges include the need to run the games on a broad range of smart phone brands and models; cross-platform code development frameworks are needed to provide cross-compilation of source code and help the developer compensate for differences in hardware performance, sensor capabilities, communications systems, operating system, and programming languages. Testing challenges include validating requirements relative to the distributed joint state of the system, which allows temporary network partitions that lead to inconsistent state views and the need to involve many to tens of different devices.

The author has implemented three multiplayer games promoting outdoor activity and social interactions; these have allowed experimentation with the concepts and provided initial trials of a Geocast Games Architecture and rapid recoherence design.

Multilevel data analytic studies are used to support user experience assessments during the development of a serious game, AGoogleADay.com, by Daniel Russell in Chapter 9. The game is a *trivia question*-style game