Chapter 1

Introducing Computer Games and Software Engineering

Kendra Cooper
The University of Texas, Dallas

Walt Scacchi
University of California, Irvine

1. The Emerging Field of Computer Games and Software Engineering

Computer games (CG) are rich, complex, and often large-scale software applications. CG are a significant, interesting, and often compelling software application domain for innovative research in software engineering (SE) techniques and technologies. Computer games are progressively changing the everyday world in many positive ways [ReR09]. Game developers, whether focusing on entertainment market opportunities or game-based applications in non-entertainment domains like education, healthcare, 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 genre include action/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/training, and massively multiplayer online games (MMOGs). This suggests that knowledge about one type of game (e.g., RPGs like *Dungeons and Dragons*) does not subsume, contain, nor provide the game play experience, player control interface, game play scenarios, or player actions found in other types of games. So being highly skilled in the art of one type of game software development (e.g., building a turn-taking RPG) does not imply the corresponding level of skill in developing another type of game software (e.g., a continuous play twitch/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 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, like card or board games, raise the obvious possibility for a single game engine to be developed and shared/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 8X8 checkerboard, though the shape and appearance of the game play pieces differs, 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). So being skilled in the art of developing a Checkers game 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 likely 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 [BEW98, Gre09]. Subsequently, developing the 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 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 multiple player game network.

2. A 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 like how best to develop computer games for play across global cultures, or through multi-site, global SE practices.

The history of techniques for CG software development go back many decades, far enough back to coincide with the emergence of SE as a field of research and practices in the late 1960's. As computer game software development was new and unfamiliar, people benefitted from publications of "open source" game software, often written in programming languages like Fortran [Spe68]. Before that, interest in computer-based playing against human opponents in popular parlor games like Chess, Checkers, Poker, Bridge, Backgammon, Go, and others was an early fascination of researchers exploring the potential of artificial intelligence using computers [Sam60]. It should be noted that these computer game efforts did not rely on graphic interfaces that 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 computer games began to capture the imagination of software developers and end-users as opening up new worlds of interactive play for human player-computer or player versus player game play to large public audiences, and later to end-user development or modification of commercial games [BCR04].

Combat oriented maze games like *Maze War*, *Amaze* [BeC85], *MiMaze* [GaD98] and others [Swe98] 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 [Bar90], along with early social media capabilities like online forums (threaded email lists), multi-user chat (including Internet Relay Chat) and online chat meeting rooms (from multi-user dungeons--MUDs), that 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, else suffer the likely 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 which 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 like *Spacewar!, Adventure, Zork*, and others using Internet-accessible file servers via file transfer protocols (FTP).

The pioneering development of DOOM in the early 1990's [Hal92, Kus03], along with the growing popularity of Internet-based file sharing, along side 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 computer games as 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 like DOOM, by offering basic means for end-user programming and modification of computer games that might have little viable commercial market sales remaining [Au02, Kus03]. The end-users' ability to therefore engage in primitive CGSE via game modding was thus set into motion [cf. Au02, BCR04, Mor03, Sca10]. Other game development studios like Epic Games also began to share their game software development tools as software development kits (SDKs) like 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 CDRom media that retail consumers would purchase starting in the late 1990's. Similarly, online sharing of game software as either retail product or free game mod, was formalized by Valve Software through their provision of *Steam* online game distribution service, along with its integrated payment services [Au02, Sca10].

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 currently eight volume series on *Game Programming Gems* published by Charles River Media and later Cengage Learning PTR (2000-2010), reveal long-standing interest on the part of game makers to view their undertaking as one primarily focused on programming rather than SE, where the 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 [FSH04, Mei03, Rog10, SaZ04, Sch08]. This points to another gap, as many students interested in making CG choose to focus their attention towards a playful user experience, while slighting or 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.

3. Topics in Computer Games and Software Engineering

This book collects and contributes eleven chapters that begin to systematically explore the CGSE space. The chapters that follow draw attention to topics such as CG and SE Education; 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

infrastructures, 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.

3.1 Computer games and software engineering education

Swartout and van Lent were among the earliest to recognize the potential of bringing CG and game-based virtual worlds into mainstream CS education and system development expertise [Swv03]. Zyda followed by further bringing attention to the challenge of how best to educate a new generation of CG developers [Zyd06]. 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), along with programs that would stress the expertise in CS required of game software developers or infrastructural systems engineers. These pioneers in CS research recognized the practical utility of CG beyond entertainment could be marshalled and directed to support serious game development for training and educational applications. However, for both of these visions for undergraduate CS education, SE plays little role 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 CS students worldwide are literate in CG and game play. Building from this insight, Oh Navarro and van der Hoek [OhV05, OvH09], the Claypools [ClC04], and Wang and students [Wan11, WOM08] were among the earliest to call out the opportunity for focusing on the incorporation CG deep into SE education (SEE) coursework.

Oh Navarro and van der Hoek started in the late 1990's exploring the innovative idea of teaching SE project dynamics through a simulation-based SE role-playing game, 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 in texts that mostly ignore modern SE principles and practices [FSH04, Mei03, Rog10, SaZ04, Sch08], may sometimes lead to projects that produce interesting, playful games, but do so with minimal demonstration of SE skill or expertise.

Wang and students have demonstrated how other CG and game play experiences can be introduced into CS or SEE coursework, through gamifying course lectures that facilitate faculty-student interactions and feedback [WOM08]. Wang [Wan11] along with Cooper and Longstreet [CoL12, and in Chapter 3] expand their visions for SEE by incorporating contemporary SE practices like software architecture and model-driven development. More broadly, Chapters 2 through 6 in this book all speak to different ways and means for advancing SEE through CG.

Finally, for those readers who teach SEE project courses, it may be valuable for students to become engaged with CGSE through exposure to the history of computer game software development, including review of the some of the pioneering papers/reports cited earlier in this introductory chapter. Similarly, whether to structure the SEE coursework projects as massively open online courses (MOOCs), or around competitive, inter-team game jams also merits consideration. Such competitions can serve as testbeds 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. Ideas like this are discussed in Chapter 12.

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 [CNS05, AmS10], much like it has been for mainstream SE. However, there are still relatively few game development approaches that employ SE requirements development methods like use cases and scenario-based design [Wal03].

Many game developers in industry have reviewed the informal game “post mortems” that first began to be published in *Game Developer* magazine in the 1990's [Gro03], and more recently on the *Gamasutra.com* online portal. Grossman's [Gro03] edited collection of fifty or so post-mortem best reveal common problem that recur in game development projects, which cluster around project software and content development scheduling, budget shifts (generally development budget cuts), and other non-functional requirements that drift or shift in importance during game development projects [AlS13, PPT09]. 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, non-functional requirements (e.g., the game should be fun to play but hard to master; game should run on 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 post-project 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.

3.3 Game software architecture design

CG 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. While 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 [SeS06, Sca10]. But other software architecture concerns exist. For instance, there are at least four kinds of CG software architecture that arise in networked multiplayer games: (a) the static and dynamic run-time

architectures for a game engine; (b) the architecture of the game development frameworks or SDKs that embed a game's development architecture together with its game engine [Wan11]; (c) the architectural distribution of software functionality and data processing services for networked multiplayer games; and (d) the informational and geographical architecture of the game levels as designed play spaces. For example, for (c) 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") [Ale03, Bar90, BeC85, BEW98, GaD98, Hal92, Swe98].

In contrast, the focus on CG as interactive media often sees little/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 are assumed will design and provide [FSH04, Mei03, Rog10, SaZ04, Sch08]. 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 finds ways to span the gap.

3.4 Game software playtesting

CG as complex software applications for potentially millions of end-users will consistently and routinely manifest bugs [LWW10]. Again, this is part of the puzzle of any complex SE effort, so games are no exception. However, as user experience (UX) and thus user satisfaction, may be key to driving viral social media that helps promote retail game sales and adoption, then paying close attention to bugs and features in CG development and usability [PWS08] may be key to the economic viability of a game development studio. Further, again from knowledge of decades of experience in developing large-scale software applications, 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 CG 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/enhanced game version (or prototype) releases. Early and ongoing game playtesting will likely come to be central facet of CGSE, as will tools and techniques for collecting, analyzing, and visualizing game playtesting data [DrC09, Zoe13]. 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. So CG developers, much like mainstream software engineers are moving towards incremental development, rapid release, and user playtesting to drive new product release versions.

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 [FSR11]. For example, many successful CG 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, Unreal Tournament 2007*). Whether

or how software product lines concepts and methods can be employed in widespread CG business models is unclear and under-explored. 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 in 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 NPCs), as perhaps its most visible form of software reuse practice. Game software development kits (SDKs), game engines, procedural game content generation tools, and game middleware services all undergoing 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 game players may recognize such games as offering derivative play experience, rather than as original play experiences. However, moving to catalogs of pattern/anti-patterns for game requirements, architecture and design patterns for game software product lines [FSR11], 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, topics like these are explored in Chapters 10 and 11.

Other approaches to software reuse may be found in free/open source software for CG development [Sca04], and also AI/computational intelligence methods for (semi-)automated content generation and level design [IEE14].

3.6 Game services and scalability infrastructures

CG range from small-scale, stand-alone applications for smart-phones (e.g., app games) to large-scale, distributed, real-time MMOGs. CG 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 [DsC09, Zoe13]. Prior knowledge of the development of multiplayer game software systems and networking services [cf. Ale03, BeC86, GaD98, Swe98] may be essential for CGSE students focusing on development of social/mobile MMOGs. In order to engage the users and promote the adoption and on-going use of such large and upward/downward scalable applications, CGSE techniques have significant potential, but require further articulation and refinement. Questions on the integration of game-play 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 [Ale03, Bar90, BeC85, GaD98, Swe98]. Whether/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 CG 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 like the *National Gaming League* for professional level game play tournaments, point to other CGSE challenges like cheat prevention, latency equalization, and statistical scoring systems, complex data analytics [DsC09] and play data visualizations [Zoe13] that support game systems that are balanced and performance (monitoring) equalized for professional-level tournaments. The social sciences could provide insight on how to attract, engage, and retain players across demographic groups (e.g., age, gender, geographic location), must like recent advances in Cooperative and Human Aspects in SE (CHASE) 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.

4. The Emergence of a Community of Interest in CGSE

At the core of CG 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 hasn't 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, they 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 SE education. 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. Chapters in this book by: Sheth, Bell, and Kaiser; and Hall were initially presented in simpler form at GAS 2011.

In the following year, 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; healthcare; fitness; sustainable activities (e.g., recycling awareness); and so on. Successful CG 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. Software engineering techniques spanning patterns (requirements, design), middleware, testing techniques, development environments and processes for building sustainable software are of great interest. Chapters in this book by Reichart, Ismailovic, Pagano and Brugge; and Dragert, Kienzle and Verbrugge 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; healthcare; 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 in this book by Russell, and also by Xie, Tillman, and colleagues were initially presented in earlier form at GAS 2013. The chapters by Wang; Debeauvais, Valadares, and Lopes; and Scacchi 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.

Finally, the topic of how best to elevate the emerging results and discipline of CGSE was surfaced and put into motion at the end of GAS 2013, of which this book is now the product of that effort. Many participants from 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 workshop, conference, or journals were reviewed for possible inclusion in this book. So 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 to this first book on CGSE. The interested reader is encouraged to consider focusing on topics of greatest interest first, but to also review the other chapters as complementary issues found at the intersection of CG and SE are covered across the set of remaining chapters.

5. Introducing the Chapters

A comprehensive literature review of CG in software education is presented in **Chapter 2** by Alf Inge Wang and Bian Wu. They explore how computer game development is being integrated into CS 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: CS (37 articles), SE (16 articles), and applied CS (13 articles). For CS, a variety of topics (e.g., programming, artificial intelligence, algorithms) are being taught at different levels (university, elementary, middle, and high school). Game development approaches in university courses on programming dominate the findings, followed by artificial intelligence. For SE, a variety of topics (e.g., architecture, object oriented analysis and design, testing) are being taught in university courses.

Game development approaches on design topics (architecture, object-oriented) lead the findings, followed by testing. For applied CS, a variety of topics (e.g., game design, game development with a focus on game design, 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 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 GDFs and languages are 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 these articles. 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 CS. Often the 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 looks of a game.

The third research question focuses on identifying common experiences from using game development to teach CS 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 the game development instead of the topic being taught. In addition many articles reported that game development positively supported recruiting and enrollment efforts in CS and SE.

Based on the results of this survey, the authors propose a set of recommendations for choosing an appropriate GDF 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, 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 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 has 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 semi-formal tailored 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 semi-formal model into formal, executable models in Statecharts and XML. A Statechart can undergo comprehensive simulation/animation to verify the model‘s behavior using existing tool support; errors can be identified and corrected both in the Statechart model and the semi-formal model as needed. The 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 that 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., 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, semi-formally, 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 role playing games, 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) Software Engineering. HALO can make the software development process and in particular, the testing process, more fun and social by using themes from popular computer games. HALO represents SE tasks as quests; a storyline binds multiple quests together. Quests can be individual, requiring a developer to work alone, or group, requiring a developer to form a team and work collaboratively towards their objective. Social rewards in HALO can include titles - prefixes or suffixes for 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 repeat 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 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 want 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.

Tao Xie, Nikolai Tillmann, Jonathan de Halleux, and Judith Bishop in **Chapter 5** 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 student then proceeds 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 into 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 adopted as a major platform for assignments in a graduate software engineering course and a coding-duel contest was recently 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 on-line programming exercise systems holds great promise as a tool in SE education.

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 two 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 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, like 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 massively multiuser online games (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 (REST) principles, which have been used very successfully for scaling web applications, with the architecture level design of MMOGs. The proposed architecture, RESTful Cient-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, 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: 1) from the proxy to the clients; and 2) from the server to the database, with and without caching. The experiments show bandwidth from the proxy to the clients increase 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 the behavior of 1) CPU utilization and round trip time (latency) as the number of clients increase; 2) CPU utilization and context switches per second as the number of clients increase; and 3) 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, development of a reference implementation, 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 it can be applied broadly, supporting the development of diverse MMOGs.

The challenges in developing multi-player outdoor smartphone games are presented in five core areas of software engineering (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 multi-player 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 smartphones, iPod Touch, etc. and they need to rely on device-to-device communication, as network access may not be available. These two characteristics have profound impacts on the 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 on top of a fully distributed architecture, subject to sporadic device communication. A collection of design issues fall under this distributed joint state problem: when devices have been out of communication, they need to re-synchronize in a rapid and fair way when they re-establish a connection. The implementation challenges include the need to run the games on a broad range of smartphone 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 multi-player 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 re-coherence design.

Multi-level data analytic studies are used to support user experience assessments during the development of a serious game, *AGoogleADay.com*, by Dan Russell in Chapter 9. The game is a "trivia question" style game that it is intended to motivate the more sophisticated use of the Google search engine. The studies are designed to understand the overall user experience by analyzing game player behavior at three different time scales: micro-, meso-, and macro-scale. The micro-scale studies measure behaviors from milliseconds up to minutes of behavior, usually with a small number of people in a lab setting. These studies provide insight into the mechanics of the game—where players look, how they perceive the game elements, and what is left unnoticed and unexplored. Eye tracking is used to understand how a player visually scans the display and the results are visualized in heat maps. A sample heat map is presented that shows a player spent most of their time towards the bottom of the display (reading the question) and near the top (at the search query).

The meso-scale studies measure behaviors from minutes to hours, observing people playing the game in natural settings. These provide insights into why people choose to play, why they stop their play, as well as understanding what makes the game engaging. The players are interviewed to acquire data in these

studies. The players have almost no learnability issues with the game – it is straightforward to learn and play. When presented with questions on unfamiliar topics, some players prefer to move on to another question; other reported greater satisfaction when they could accomplish the task on their own, without any hints of the solution. These results lead to the introduction of a "Skip" feature and a modification to the existing "Clue" feature in the game.

The macro-scale measures behaviors from days to weeks and months. Typically, this involves large numbers of players, and is usually an analysis of the logs of many people playing the game over time. These studies reveal unexpected behaviors, such as the cumulative effect of people returning to the game, and then backing up to play previous days' games; in addition a significant drop in participation is identified and traced to an error in advertising the game.

Overall, Russell's approach of understanding complex user behavior at three different time scales, using three different kinds of studies, provides the community with a broadly applicable framework for assessing software systems with complex user interfaces.

A formal, structured approach to effectively re-use artificial intelligence (AI) components in game development to create interesting behavior is presented by Christopher Dragert, Jörg Kienzle, and Clark Verbrugge in **Chapter 10**. The approach is based on a layered Statechart, which provides inherent modularity with nesting capabilities. Each Statechart defines a single behavioural concern, such as sensing data. The individual Statecharts are used to create more complex, higher level groups of intelligent behavior; the exhibited behaviour is a function of the superposition of states. Under this model the lowest layer contains *sensors*, which read events created as the game state changes. Events are passed up to *analysers* that interpret and combine diverse data from sensors to form a coherent view of the game state. The next layer contains *memorizers* that store analyzed data and complex state information. The highest layer is the *strategic decider*, which reacts to analysed and memorized data and chooses a high level goal. The high level goal triggers a *tactical decider* to determine how it will be executed. Ultimately, *executors* contained in another layer enact decisions in order to translate goals into actions. Depending on the current state of the NPC, certain commands can cause conflicts or suboptimal courses of action, which are corrected by *coordinators*. The final layer contains *actuators*, which modify the game-state.

The re-use approach is illustrated by creating a garbage collector NPC through the reuse of large portions of the AI designed for a squirrel NPC. By treating the AI as a collection of interacting modules, many behavioural similarities emerge; two-thirds of the AI modules in the garbage collector are reused from the squirrel. The authors have also created tool support to demonstrate the practical potential of their approach. The tool directs and facilitates the development process, taking in Statecharts and associated classes, providing an interface for producing novel AIs from a library of behaviours, and exporting code that can be directly incorporated into a game. By formally representing and understanding game code associated with specific behaviours, the constructed AI can be rigorously analyzed, ensuring code and functional group dependencies are properly satisfied.

A collection of five case studies exploring the issue of reuse in game play mechanics from a repurposing perspective on the design of computer games and game-based virtual worlds is presented by Walt Scacchi in **Chapter 11**. The case studies are research and development projects that have been created

with diverse purposes, target audiences, and external collaborators over an extended period of time. The projects include a K-6 educational game on informal life science education, training for technicians in advanced, semi-conductor manufacturing, training for newly hired personnel on business processes, training for state employees on sexual harassment, multi-player mission planning, and games on science mission research projects. In their creation, designers have drawn upon design level, game play mechanics to reuse from existing games; here the collection of case studies is analyzed from this repurposing perspective. For example, the first case study focuses on games for students in grades K-6 on informal life science education. The games were on dinosaurs and their ecology, encompassing fundamental relationships that evolve over time including prey-predator food chains and the "circle of life" (mulch contributes to the plants, plants to herbivores, herbivores to carnivores, and carnivores to mulch). The design of the game play mechanics to highlight these relationships was drawn from tile-matching puzzle games, *Tetris* and *Dr. Mario*, where game play is motivated by time: the patterns fall into and progress across the playing field at a certain pace. In the new game, tiles for mulch, plants, herbivores, and carnivores are used. When carnivore (predators) align with herbivores (prey) they form a simple food chain, allowing the survival of the carnivore and the consumption of the prey.

The analysis of these case studies identifies five ways to repurpose game play mechanics: (a) appropriating play mechanics from functional similar games, as briefly described above; (b) modding game play levels, characters, and weapons through a game-specific software development kit; (c) substituting in-game character/non-player character dialogs along with adopting multi-player role-play scenarios; (d) employing play mechanisms for resource allocation with uncertainty that re-enact classic game theory problems; and (e) identifying game design patterns (quests, multiplayer role-playing) that can be used to develop families of games across science research problem-solving domains.

This study provides the community with a foundation to systematically observe, identify, conceptually extract, generalize and then specialize/tailor game play mechanics that are provided in existing games. Such study can also inform how best to design or generate game content assets (e.g., non-player characters (NPCs), behavioral objects, extensible rule sets for modifying game play mechanics) for modding or repurposing, along with how software domain analysis and modeling techniques can support them.

Finally, in **Chapter 12**, we review the follow-on ideas and suggestions identified in preceding chapters to serve as an outline for identifying an agenda for the future of research in CGSE.

6. Summary

This chapter serves as an introduction to this first collection of research papers in the area of Computer Games and Software Engineering. It started with a brief recapitulation of the history of computer game software development that helps set the stage for the introduction of the research contributions in the chapters that follow. The next chapters are the core of this book, which is followed by a brief outlook on the future of research in computer game and software engineering, as both are identified within the contributing chapters, as well as with respect to some of the grand challenges in software engineering, and other complementary R&D opportunities for CGSE.

7. Acknowledgements

The research of Walt Scacchi was supported by grants #0808783, #1041918, and #1256593 from the U.S. National Science Foundation. No review, approval, or endorsement is implied. Only the authors of this Chapter are responsible for statements and recommendations made herein.

8. References

[Ale03] Alexander, T. (Ed.) (2003). *Massively Multiplayer Game Development*, Charles River Media, Hingham, MA.

[AmS10] Ampatzoglou, A. and Stamelos, I. (2010). Software engineering research for computer games: A systematic review. *Information and Software Technology* 52(9), September, 888-901.

[Au02] Au, J.W. (2002). Triumph of the Mod. *Salon*. 16 April 2002. http://www.salon.com/tech/feature/2002/04/16/modding/

[Bar90] Bartle, R. (1990). *Interactive multi-user computer games*. Technical report, British Telecom, June 1990, http://www.mud.co.uk/richard/imucg.htm

[BeC85] Berglund, E.J. and Cheriton, D. Amaze: A multiplayer computer game, *IEEE Software,* 2, 30-39, May 1985.

[BEW98] Bishop, L., Eberley, D., Whitted, T., Finch, M., and Shantz, M., Designing a PC game engine, *IEEE Computer Graphics and Applications*, January-February, 46-53, 1998.

[BCR04] Burnett, M., Cook, C., and Rothermel, G. (2004). End-user software engineering, *Communications ACM*, 47(9), 53-58.

[CNS05] Callele, D., Neufeld, E., and Schneider, K. (2005). Requirements Engineering and the Creative Process in the Video Game Industry, *Proc. 13th Intern. Conf. Requirements Engineering,* (RE'05) 240-250.

[ClC04] Claypool, K. and Claypool, M. (2005). Teaching Software Engineering through Game Design, in *Proc. 10th SIGCSE Conf. Innovation and Technology in Computer Science Education* (ITiCSE '05), pp. 123–127, Caparica, Portugal.

[CoL12] Cooper, K. and Longstreet, S. (2012). Towards Model-driven Game Engineering for Serious Educational Games: Tailored Use Cases for Game Requirements, *17th. Intern. Conf., Computer Game*s, IEEE Computer Society, 208-212.

[DrC09] Drachen, A. and Canossa, A. (2009). Towards gameplay analysis via gameplay metrics, in *Proc. Intern. MindTrek Conference*, 202–209.

[FSH04] Fullerton, T., Swain, C., Hoffman, S. (2004). *Game Design Workshop: Designing, Prototyping and Playtesting Games*. CMP Books, February 2004.

[FSR11] Furtado, A W B, Santos, A L M, Ramalho, G.L., de Almeida, E.S. (2011). Improving Digital Game Development with Software Product Lines, *IEEE Software,* 28(5), 30-37, September-October.

[GaD98] Gautier, L. and Dior, C., Design and Evaluation of MiMaze, a Multi-player Game on the Internet, in *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, 233-236, 1998.

[Gre09] Gregory, J. (2009). *Game Engine Architecture*. AK Peters/CRC Press, Boca Raton, FL.

[Gro03] Grossman, A. (Ed.) (2003). *Postmortems from Game Developer: Insights from the Developers of Unreal Tournament, Black and White, Age of Empires, and Other Top-Selling Games*, Focal Press.

[Hal92] Hall, T. (1992). *DOOM Bible*, Revision Number .02, ID Software, Austin, TX, 11 November 1992. http://5years.doomworld.com/doombible/doombible.pdf

[IEE14] IEEE (2014). *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE Computational Intelligence Society.

[Kus03] Kushner, D., *Masters of Doom: How two guys created an empire and transformed pop culture*, Random House, New York.

[LWW10] Lewis, C., Whitehead, J. and Wardrip-Fruin, N. (2010). What went wrong: a taxonomy of video game bugs. In *Proc. Fifth Intern. Conf. Foundations of Digital Games* (FDG '10). ACM, New York, NY, USA, 108-115.

[Mei03] Meigs, T. (2003). *Ultimate Game Design: Building Game Worlds*, McGraw-Hill, New York.

[Mor03] Morris, S. (2003). WADs, bots, and mods: Multiplayer FPS games and co–creative media, *Level Up Conference Proceedings: 2003 Digital Games Research Association Conference* (Utrecht). Online at: https://web.archive.org/web/20120627070927/http://www.digra.org/dl/db/05150.21522

[OhV05] Oh Navarro, E. & Van der Hoek, A. (2005). Software process modeling for an educational software engineering simulation game. *Software Process Improvement and Practice*, 10(3), 311–325.

[OhV09] Oh Navarro, E. & Van der Hoek, A. (2009). Multi-site evaluation of SimSE. *SIGCSE Bulletin,* 41(1), March, 326-330.

[PPT09] Petrillo, F., Pimenta, M., Trindade, F., and Dietrich, C. (2009). What went wrong? A survey of problems in game development. *Computers in Entertainment,* 7(1), Article 13, February, DOI=10.1145/1486508.1486521

[PWS08] Pinelle, D., Wong, N., and Stach, T. (2008). Heuristic evaluation for games: usability principles for video game design. In *Proc. SIGCHI Conference on Human Factors in Computing Systems* (CHI '08). ACM, New York, NY, USA, 1453-1462.

[ReR09] Reeves, B. and Read, J.L. (2009). *Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete*, Harvard Business Press, Boston, MA.

[Rog10] Rogers, S. (2010). *Level Up!: The Guide to Great Video Game Design*, Wiley, New York.

[SaZ04] Salen, K. and Zimmerman, E. (2004). *Rules of Play: Game Design Fundamentals*, MIT Press, Cambridge, MA.

[Sam60] Samuel, A. (1960). Programming computers to play games. In F. Alt (Ed.) *Advances in Computers*, Vol. 1, 165-192, Academic Press, New York.

[Sca04] Scacchi, W. (2004). Free/Open Source Software Development Practices in the Game Community, *IEEE Software*, 21(1), 59-67, January/February.

[Sca10] Scacchi, W. (2010). Computer Game Mods, Modders, Modding, and the Mod Scene, *First Monday*, 15(5), May 2010. http://firstmonday.org/ojs/index.php/fm/article/view/2965/2526

[SNA08] Scacchi, W., Nideffer, R. and Adams, J. (2008). Collaborative Game Environments for Informal Science Education: DinoQuest and DinoQuest Online, *Proc. IEEE Conf. Collaboration Technology and Systems,* (CTS 2008), Irvine, CA 229-236.

[Sch08] Schell, J. (2008). *The Art of Game Design: A book of lenses*, Morgan Kauffman Elsevier, Burlington, MA.

[SeS06] Seif El-Nasr, M. and Smith, B.K. (2006). Learning through game modding. *Computers in Entertainment,* 4(1), Article 7, January. DOI=10.1145/1111293.1111301

[Spe68] Spencer, D.D. (1968). *Game Playing with Computers*, Spartan Books, Macmillan & Co. Ltd., New York.

[Swv03] Swartout, W. and van Lent, M. (2003). Making a game of system design. *Communications ACM,* 46(7), 32-39.

[Swe98] Sweeney, T. (1998). *Unreal Networking Architecture*, Epic MegaGames Inc., 21 July 1999. https://web.archive.org/web/20100728233924/http://unreal.epicgames.com/Network.htm

[Wal03] Walker, M. (2003). Describing Game Behavior with Use Cases, in Alexander, T. (Ed.), *Massively Multiplayer Game Development*, 49-70. Charles River Media, Hingham, MA.

[WaK13] Wallner, G., and S. Kriglstein. Visualization-Based Analysis of Gameplay Data-A Review of Literature. *Entertainment Computing,* 4(3), 143-155, (2013).

[Wan11] Wang, A.I. (2011). Extensive Evaluation of Using a Game Project in a Software Architecture Course, *ACM Trans. Computing Education* (TOCE), 11(1), 1-28.

[WOM08] Wang, A.I., Ofsdahl, T., and Morch-Storstein, O.K. (2008). An Evaluation of a Mobile Game Concept for Lectures. In *Proc. 21st Conf. Software Engineering Education and Training* (CSEET '08). IEEE Computer Society, Washington, DC, USA, 197-204.

[Zoe13] Zoeller, G. (2013). Game Development Telemetry in Production, in Seif El-Nasr, M., Drachen, A., and Canossa, A. (Eds.), *Game Analytics: Maximizing the Value of Player Data*, Springer Verlag, New York, 111-135.

[Zyd06] Zyda, M. (2006). Educating the Next Generation of Game Developers, *Computer*, 39(6), 30-34.