Research Challenges at the Intersection of Computer Games and Software Engineering

Walt Scacchi
Institute for Virtual Environments and Computer Games, and
Institute for Software Research
University of California, Irvine
wscacchi@ics.uci.edu

Kendra M. Cooper
Computer Science Department
University of Texas, Dallas
kendra.m.cooper@gmail.com

ABSTRACT

This paper provides an overview and review of R&D studies, findings, and practices that identify important problems that constitute an emerging program of future R&D opportunities relevant to established scholars and new students interested in computer games and software engineering (CGSE). This includes examining how computer games may be used to address long-standing, grand challenge problems in software engineering in new ways. The review also examines other challenges in game software requirements engineering, game software design, game software testing, teamwork processes in CGSE, global CGSE, and other closely related areas for CGSE research. From these results, it becomes possible to identify and outline possible areas for future CGSE research opportunities that may be appropriate for consideration by students, scholars, or research agencies seeking to build up scientific and technological capabilities in CGSE research and educational practice.

Keywords

Computer Games, Software Engineering, Research, Grand Challenges, Software Engineering Education

1. INTRODUCTION

Over the past five years, 2011-2015, there have been four international workshops hosted at the International Conference on Software Engineering that focus on emerging research in Games and Software (GAS). The 2015 Workshop is scheduled for May in Florence, Italy (cf. http://2015.gasworkshop.org/). Dozens of researchers, spanning established senior scholars with long research legacies to graduate students early in their research careers and studies, have participated in the workshops. Along the way, dozens of research papers have been submitted, and smaller numbers have been selected for presentation and publication in electronic proceedings hosted by ACM and IEEE Computer Society. The authors have participated in these workshops as paper authors, invited speakers, workshop organizers, and program co-chairs. In recognition of this emerging area of research that spans the communities of computer game developers and software engineers, it has been possible to develop a perspective about many categories of R&D problems that lie at the intersection of these two distinct communities. Subsequently, it has been possible to identify and re-examine some of the grand challenges in software engineering to see how they are manifest in this cross-disciplinary intersection. The purpose of this paper is therefore to provide an overview and review of studies, findings, and practices that identify important problems, and future R&D opportunities, in computer games and software engineering (CGSE), that have been informed by the collection of papers and presentations at the GAS Workshops. Overall, there are many possible futures for research in CGSE.

Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015), June 22-25, 2015, Pacific Grove, CA, USA.
ISBN 978-0-9913982-4-9.

Computer games may well be the quintessential domain for Computer Science and SE R&D. Why? Modern multi-player online games (MMOG) must address core issues in just about every major area of CS research and education. Such games entail the development, integration, and balancing of software capabilities drawn from algorithm design and complexity, artificial intelligence, computer graphics, computer-supported cooperative work/play, database management systems, human-computer interaction and interface design, operating systems and resource/storage management, networking, programming or scripting language design and interpretation, performance monitoring, and more. Few other software system application arenas demand such technical mastery and integration skill. Yet game development is expected to rely on such mastery, and provide a game play experience that most users find satisfying, fun, and engaging. Computer games are thus an excellent domain for which to research and develop new ways and means for (game) software engineering.

Future R&D opportunities in CGSE are emerging at the intersections of CG and SE with each other, and with other disciplines. A starting question to ask is, how CG development different from SE [18]? More broadly, an overall question for the student or professional reading this paper is *how will future computer games succeed/fail with/without SE tools, techniques, and concepts?* For example, if SE is mostly irrelevant to the successful development, deployment, and sustained evolution of computer games, why is this so? Is (a) traditional SE no longer relevant overall to upcoming generations of game software developers, or is (b) there something about the ways and means by which computer games, game development projects, and game development organizations operate and engage players that eludes recognition by SE researchers and educators? While the former (a) seems unlikely and unpopular to contemplate, the latter (b) points to opportunities for future research in SE. The challenges are thus yours to consider and address.

The mainstream CG industry is a global endeavor with multi-billion dollar game development companies, as well as untold numbers of independent small-to-mid sized game studios that

make games for commercial markets or specialty applications (e.g., game-based training). CG are the most popular type of application on mobile devices, based on downloads and time spent using such apps. So what is going on with game development, and what role can or should SE play in advancing the commercial interests of game development firms? What are the technical SE challenges facing game developers? What are the best ways/means to engineer fun-filled experience and playful challenges that millions of players world-wide and across cultures want through game software?

With questions like this in mind, this paper identifies some future arenas and issues for research found at the intersections of CGSE. First, we look to long-term challenges in SE that may be addressed through the lens of CGSE, to see if there is something new or different to be learned that may advance our understanding of how to better address such challenges. Following this, we look to other grand challenge areas within science and engineering arenas that span global markets, to help see if there are potential opportunities for students and scholars in different geographic regions that may be specific to their industrial workforce, external research funding bases, and emerging game market opportunities. This last topic is intended to help broaden the perspective of seeing the future of research in CGSE is not just something relevant to the U.S., but is in fact a global opportunity area that can be advanced both scientifically and socio-economically through CGSE. As such, we turn to examine opportunities for future CGSE research that arise in mainstream SE.

2. GRAND CHALENGES IN SOFTWARE ENGINEERING THROUGH COMPUTER GAMES

Many traditional grand challenges in SE arise during the development of computer games as complex software systems. This includes using game to solve challenge problems in large-scale software engineering, game software requirements engineering, game software design and testing, teamwork processes in game software development, global software development and global game development, and other problem areas for future research study. This set of challenges is intended to be suggestive rather than exhaustive or prioritized by importance. Each is examined in turn.

2.1 Using games to solve challenge problems in large-scale software engineering

Engineering large-scale software systems has long been recognized as a difficult problem in Computer Science. Industry reports sometimes indicate that as few as one out of three large software projects can be considered successful, while another third are outright failures, and the remainder are plagued with flaws that demand excessive maintenance or provide a modest/poor level of user satisfaction. It seems unlikely that large-scale games should be any different at least until it can be empirically substantiated whether/how game software development is fundamentally different. CG are still software applications, so their development is likely similar to other commercial software development efforts, unless evidence shows otherwise.

Engineering scalable software systems is often bracketed as its own family of challenges. These challenges similarly appear in the development of computer games like MMOG, where millions of user-players are anticipated, with thousands playing concurrently any time of day. The CGSE challenges for massively scalable games include how best to: (a) design and organize teamwork processes for game development, playtesting and post-deployment support; (b) divide the CGSE work among developers with different skills across different development sites; (c) organize and enact multi-site game software project management; design scalable game services and supporting infrastructures that may multiple host server sites; (d) manage concurrent system configurations and product release versions for multiple end-user platforms; (e) manage and secure game-based intellectual property (IP) assets and user-created content; (f) and more. Conversely, for MMOG clients, how are remote game-based middleware services to scaled down for delivery of real-time game play experience that can fit within the game's client software deployment? In short, research that informs large-scale CGSE projects, as well as scalable game system architectures for integrating remote services, represent an interesting set of challenges likely to be of great interest to game development firms.

Beyond scalability, the world of SE is frequently characterized in modern textbooks as a series of software life cycle engineering tasks. These SE tasks must be performed systematically, with care, as well as monitored and measured as a continuously improvable set of processes. That's what we teach and what we encourage students and industrial professionals to learn and practice. But many hard and difficult problems arise along the way in many SE projects while performing such tasks, and thus the mainstream SE research community focuses much research attention to advancing knowledge about tools and techniques for addressing or resolving the challenges that arise. As such, we turn to briefly explore other grand challenges of SE, along with modest enumerations of CGSE research opportunities that may follow. The interested reader can undoubtedly identify other R&D opportunities as well.

2.2 Game software requirements engineering

Software requirements engineering has been a long-term challenge in SE. It has its own international research conferences and dedicated journals that are intended to complement mainstream research in the field of SE. The existence of both the conferences and journals suggest that international interest in the research and practice in how best to elicit and articulate software functional requirements that may be formalized, as well as what to do with non-functional software requirements, is widespread. However, with computer game software, it appears that the practice of game development focuses primary attention on creating and satisfying non-functional requirements (NFRs) for the game as product, versus the game software's functional requirements which are often tacit and undocumented, or specified in hindsight [1]. NFRs like "the game must be fun to play" by a target audience of users/players (e.g., "played by males 18-35 years old" or "be gender neutral") on a target platform ("for play on Microsoft *Xbox One* consoles," "browser based," "integrated with social media sites like Facebook" or "mobile devices running iOS or Android operating systems") at some retail price point or monetization scheme ("free to play, with game-based micro-transactions using pre-paid user debit accounts" or monthly subscriptions for user accounts) is much more common [5]. The SE challenge that thus appears is determining what to do during game software development to address or satisfy such NFRs as engineering tasks. Alternatively, challenges also exist for

finding how best to gamify the requirements engineering and other SE activities [8,11].

Game developers have learned that merely providing a game whose software functions as intended (i.e., meets its functional requirements), is much less relevant than providing a game that game playtesting [12,32] empirically demonstrates satisfying diverse game player focus groups, rather than in-house game developers. In this regard, NFRs for games are testable for playability or user experience (UX), but these concerns tend not to be clearly aligned with the interests of SE testing research. Instead, it is not surprising to see industry shifting from focus on large multi-million dollar budget AAA game development projects, towards games that can be incrementally developed and released with a minimum set of game play features that can adaptively be grown to meet informal NFRs. Early success of a new minimal game with players (e.g., online player reviews, user-created game play videos, game micro-transaction rates of return) will determine whether more features will be added or integrated, thereby realizing that the game system is being developed as an emerging online interactive gameplay service, rather than just as a software product. Whether such game development prescribes "features" as subject to functional or non-functional requirements is at present an open problem in CGSE, as is the overall game development process that is driven by features that follow from incremental deployment of testable game feature sets. This is not the same as SCRUM or agile development methods, at least as they have been prescribed in textbooks, but they may represent some linkage to such practices. So investigating and better understanding the virtues and vices of new game software development practices is an open area for future CGSE research.

Another observation on requirements for game development arises with recognition that some game developers are committed end-users of the games they play [26]. Compared to traditional SE project situations where developers and users are distinct groups [1], elicitation of functional requirements can be less complicated when users are developers. This is primarily the situation for *game modders*, people who intentionally act to modify an existing game, where the released game includes access to a software development kit (SDK) that is pre-configured to afford certain types of changes to the game's appearance, rules of play, play experience or game purpose [25]. As modders act as both developer of (new) game functionality, as well as an end-user to be satisfied, there may be no communication/interest gap between developers and users. Thus, there may be no need for traditional software requirements elicitation or specification, and instead the developer-user can iteratively explore, try-out, and rapidly adapt their needs to what they can accomplish. Therefore, what is the role or value of requirements engineering in support of game modding as a development modality, is a question for further investigation and empirical exploration.

2.3 Game software design

Another classic challenge for SE is determining how best to design a software system to satisfy its requirements or specifications. As noted, computer game requirements may be dominated by NFRs, rather than functional specifications that can be formalized, verified, and validated. Thus game software design may need to account for such bias or focus.

For students and would-be game developers, many textbooks and practitioner guides target *game* design [e.g., 12,16,21,32], rather than *game software* design. Whether such texts should be considered relevant to game software design may hinge on the expectation as to whether prior software design concepts, tools and techniques are employed or ignored. For example, game design texts often focus attention on how to address NFRs for the appearance and animated "emotional" behavior of in-game characters, choice of game mechanics well-suited for the game's genre, the look and feel of game level or world design, user interface design and overlay, in-game artwork asset development and fix-up of technical art mismatches, etc. As such, these texts say little about the game's software functional requirements. This means that alternatives about game software architecture are unclear. Thus architectural design of games, and how to make trade-offs therein, remains an open challenge.

It appears that game designers are assumed or directed to employ a game SDK or development framework (e.g., *Construct 2, GameMaker: Studio, GameSalad, Microsoft XNA Game Studio, Project Spark, PyGame, Steam, Unity, Unreal Development Kit*) to realize their game design. However, as many have observed, the selection of a game development environment constrains or pre-determines what kind of computer game may be more readily developed (e.g., many SDKs primarily support development of 2D games, but not 3D games; some SDKs feature drag-and-drop game development with little ability to include new source code). Thus the game development environment encroaches into the functional requirements or NFRs space. This side steps the engineering of game software design, and instead replaces it with a development paradigm focused on game design as found in textbooks, rather than on software system design as found in SE textbooks. Similarly, if an envisioned game to be developed requires unfamiliar or unprecedented game functionality, it may be necessary to develop a game run-time environment (game engine as a software system), as well as one/more games that operate with the new game engine.

Next, as multi-player online games are complex software systems, then we can consider how the underlying software architecture facilitates or constrains how the game can be initially designed and configured, built from existing components, as well as further developed, modded, or evolved. Games are often configured using different middleware libraries or external service components that must be integrated with new game software functions, features or capabilities. However, commercial multiplayer games may incorporate twenty or more libraries of external services including user registration, micro-transaction payment, anti-cheating security, in-game or around-game online chat, behavioral character and non-player character artificial intelligence, database management, and others. Further, as some of these capabilities become further specialized, their software gives rise to their own autonomous architectural representation. Thus, the challenge arises for how best to mix, match, or reuse heterogeneous game software sub-system architectures in ways that are open to component replacement, architectural reconfiguration, multi-version run-time platforms, and even intellectual property license changes [29,30].

Another matter that arises in game software design is the development and review of game design documents. Oftentimes, the game design document is what is first produced, along with samples of in-game characters, play levels or worlds, narrative storyline for the game (if relevant), description of game play mechanics, embedded challenges or puzzles for players to overcome, and intended game play experience. As before, most of these items do not naturally map into an interconnected configuration of functional system modules, but may be more suggestive of feature-based or event-driven systems. Investigating

how game design documents compare to software design documents would be valuable, especially as new generations of students are often eager to engage in game development [6]. Similar studies may also investigate the role of other online development artifacts widely used in free/open source software (FOSS) development projects, including online chat transcripts and social media where game/software design rationales are captured, that game developers employ to support their game development projects [10,13,22,23].

2.4 Game software testing

Given the emerging story of how the development of new computer games will require new methods and technologies for game software requirements and design engineering, so to we should expect game software testing to evolve and adapt. For example, game "playtesting" focuses on the playability of a new game, along with what affects the fun of its play experiences, thereby crossing testing of game software NFRs and quality assurance with human-computer interaction and user experience (UX)—an under-explored composition of CGSE interests. Alternatively, we might look for new ways and means for engaging software developers in peer-to-peer software coding-testing duels [41]. Such approaches might be further explored in the context of software module integration or configuration management game challenges, that in turn might automatically evaluate whether the resulting system can be automatically built, packaged for remote installation, and regression tested. The Darpa Verigames initiative (described below), explores the potential for crowdsourcing approaches to testing problems found in formal software verification. But there is still much to be explored through software testing supporting CG development efforts.

Consider the challenge of addressing how best to validate NFRs for games, like "the game must be fun to play," or "operate on PCs, consoles, and mobile devices." Assessing a game's initial playability or players experience of fun, can entail eliciting end-users opinions, feedback, or concerns about their UX with current game features or capabilities [18]. In contrast, validating that a given game can operate on different platforms is a much more operational concept. Thus, what is to be determined is what game play functionality is common for focused testing, while other functionality like user interface controls may be specific to the play device platform.

An informed game software design may specify where such platform-specific customizations are to be made through functional factoring, so such design can inform subsequent testing. Formal software architectural designs may also help facilitate automated testing, as may other techniques, in ways that merit careful empirical study [18]. Additionally, how is game play UX to be assessed across platforms, since platform-specific specializations may (or not) affect the end-users play experience. Perhaps game software components and integrated configurations may be subjected to system integration testing duels. Overall, these options point to the need for one or more alternative schemes for a game software testing and playtesting hybrid regime that can assess common versus platform-specific game functionality, features and UX, as well as other NFRs.

The last area to address concerns recent interest in investigating whether computer games can be used to solve difficult problems in software verification. In the U.S., the Defense Advanced Research Project Agency (Darpa) has initiated a research program (known as *Verigames*) to see what advances can be realized through games designed for play by large distributed and loosely coordinated "crowds" of players. An initial set of games has been developed and deployed on the Web (and in one case, exclusively for a mobile platform like the Apple iPad)-- see Verigames Project Site [39]. Each game takes on an approach that decomposes problems of software verification into simpler ones, such as proving loop invariance properties [14]. Collectively, these crowdsourced games are envisioned as a fundamentally new approach to the long-standing problem of formal software verification. Of course, the games themselves must also be playable, interesting, and fun to external game players who have little/no interest in software verification challenges. Said differently, a game that satisfies its functional requirements (e.g., implements a class of loop invariance testing play mechanics) does not necessarily make the game interesting and fun to external users. Conversely, an interesting and fun game that introduces a new game mechanic, such as for determining loop invariants, might itself be an innovation in computer games, whether it does or does not solve practical problems in software verification. Nonetheless, the effort to verify large software system elements using a crowdsource approach like Verigames is a bold research undertaking near the center of future CGSE R&D.

2.5 Teamwork processes and game jams in CGSE

Many students and independent game developers participate in computer game development competitions or "game jams" [19,20,27,35,42]. These jams usually focus on clean-sheet production of a playable game usually in a limited time frame, like 24 or 48 hours, though shorter and longer competitions have been engaged. Sometimes these jams have external for-profit or non-profit sponsors, who in turn may offer financial or technology product rewards. Other times, jams offer no tangible rewards, but instead focus on going "for the win," résumé building, and shared learning experience as the desired outcome. Game jams may also be located in academic settings, so that both *intra-mural* (within school) and *inter-mural* (across schools) game jams can be undertaken in ways that complement SE Education [6,7,27,35,40].

An interesting set of research questions arises associated with game jams, perhaps most relevant to empirical studies of alternative game/software development processes, practices, methods or tools used. For example, within intra-mural game jams, it may be possible to structure and balance the game development teams by team size, game developer role and SE skill level from students at hand. Students can indicate their skill level and developer role preference, then have participants be randomly assigned to teams in ways that balance team size, role and skill level. This can mitigate against pre-formed teams with established collaborators, high skill distribution, and relatively mature game development capabilities.

Short duration jams mitigate against the consequences of team failure or participant drop out, and instead make these events more of a CGSE learning experience. In this way, in addition to focusing on game production, the overall game jam serves as a "field site" where selected CG design, SE processes and technologies [e.g., 19,42] can be comparatively investigated, following empirical SE approaches introduced 25-30 years ago [cf. 3,4]. Such field sites allow for careful empirical study of teams using a new game SDK or development technique (e.g., SCRUM, agile development, or game modding) vs. those who do not; or those who produce traditional SE documents (requirements specifications, architectural designs, test plans) and follow SE processes for their game vs. those who just focus on game design

methods. Intra-mural game jams may therefore be well-suited for longer durations (e.g., 1-2 weeks). These jams may stress short duration and co-location, along with targeted game production on a topic that is announced at the beginning of the competition. Inter-mural or open participation game jams may not be so readily structured or balanced at little cost, but instead may address other CGSE questions that better match their natural field organization and project heterogeneity.

More generally, game jams offer the opportunity to organize, design, and conduct empirical studies in CGSE that can inform both new game design practices or processes, as well as new SE practices and technologies [9]. These jams can be used to address CGSE research questions in ways underutilized in SE research. Ultimately, this can mean that SE can be viewed as a competitive team-based sport activity that can be fun for students, as well as structured to support careful empirical study [27], rather than SE being a business endeavor to produce application systems hosted on back-end infrastructures accompanied by documents that few will ever read. It also suggests that game jams may be designed as a kind of *meta-game*, so as to structure the outcomes (i.e., the games produced) to embody certain functional features, or the game production process to reward accumulative levels of progress achieved or skills mastered by different teams ("leveling up"), rather than leading to winning and losing teams.

2.6 Global software development and global CGSE

Global software development (GSD) focuses attention to challenges that arise when large software development projects are distributed in geographic space, or across many time zones. GSD often involves software development tasks being performed by teams that are culturally diverse, so that many developers will not share a first native language, but often may rely on local SE teamwork activities conducted in one's native language, while inter-site development activities and project management may be conducted and coordinated in another language (most often English). Yet many large-budget AAA games are developed in multiple international sites, or for use in global markets with different user language cultures. This implies not just the need for internationalization (or localization) of game user interfaces, but also for international game production management. It should therefore be unsurprising that culturally grounded misunderstandings and miscommunications arise during game software development, much like they do in other GSD projects.

While business efforts to reduce software development costs (e.g., through offshoring certain tasks to lower cost software labor markets) are being tried both with traditional software applications and CG, it is also apparent that critical product design feature choices or system integration activities are closely held by the lead development organization. So some software product knowledge is purposely withheld in proprietary GSD projects from global collaborators. In contrast, free/open source software (FOSS) development of commercial games is not (yet) widely practiced [22], so it is unclear whether such approaches [13,23] can overcome the challenges found in proprietary GSD projects [37]. In sum, most of this is still poorly understood from a research perspective, and few tools and techniques are being investigated to address cultural challenges in GSD, yet alone in Global CGSE.

2.7 Game-based software engineering education

There are many different ways and means for engineering of CG software systems, and some of these are now a major focus for software engineering education (SEE). To no surprise, we find that the other challenges for CGSE identified in Sections 2.1 to 2.6 are appropriate candidates for exploration through SEE project coursework.

Engineering CG software requirements, (architectural) design, testing (including UX playtesting and coding duels), teamwork processes and global CG software development are all open for application and experimentation by SEE innovators. Even large-scale CGSE may be explored through SEE, though this will demand additional expertise by SE educators in knowing how to mobilize open software infrastructural services for use within academic SEE coursework.

SEE project coursework may be organized and structured about team-based game development competitions. The goal of such competitions is not so much determination of winners and losers, but of creating CG development challenges that require, incentivize, or reward the use of SE methods for CG requirements, design, and testing using game SDKs and/or remote middleware services. Similarly, accommodating game development competitions that elicit submissions from international or globally dispersed SEE student project teams might therefore also focus comparative assessment on how well student teams practice and demonstrate GSD concepts and techniques through their CG development efforts.

Finally, other areas of SE research and practice, such as configuration management, build and release management, reverse engineering, FOSS development, agile development, classroom coursework [34], and more may all be seen as potential candidates for gamification in ways that may be engaged through SEE coursework and team projects.

3. OTHER RESEARCH OPPORTUNITY AREAS FOR CGSE

There are of course many other areas and topics that lie at the intersection of CGSE that do not specifically target grand challenge problems in SE, but that represent substantial communities of interest from a CGSE perspective. These include automated generations of games, could-based game services and infrastructure, and game software repositories and data management services.

3.1 Automated generation of computer games

One area of CGSE research interest not addressed above focuses on the creation of new technologies for (semi-) automated generation of computer games [cf. 28]. This kind of research may focus on the invention of new concepts, techniques, or tools for the generation of ready-to-play games based on specifications of game play narrative (like feature film scripts or screenplays), emergent agent-based interaction rules, or functional game software specifications. Such efforts represent a reconstitution of automatic programming or knowledge-based software development approaches once avidly explored in SE research, but now displaced by more conventional programming-centric approaches that average programmers can perform. While there is

an active community of researchers working on computational intelligence, agent-based systems, and procedurally generated worlds for games, much of these efforts are not (yet) well addressed from a CGSE perspective. For example, current efforts at procedural generation of game worlds often focuses attention to the generation of visual in-game object models or composed levels, such as automatically generated trees/vegetation, buildings, building interiors, cityscapes, waterways and geographic terrains [36,38]. However, these efforts do not generate in-game characters with complex social behaviors in different roles that operate within socio-political cultural systems that may/not feature emergent competitive or combative dynamics, nor utilize engines that provide complex physical or ecosystem simulations (e.g., processes like digestion, reproduction, speciation, or ecosystem evolution) at different physical scales (from Nano to human scale, or from human to cosmological scale) [2,24,28]. Thus much innovation is possible in this area.

3.2 Cloud-based game software services

In other research, there is growing interest in determining whether or how best to employ and engineer cloud-based computing infrastructure and services to support online games [13,17]. Cloud-based software system architectures are recognized to offer scalable services, especially for remote storage, file sharing and content distribution, commercial (payment) transactions, and remote computation among others. These architectures may inform the deployment of CG where such scalability is relevant to game publishing business ventures. However, one nagging challenge for CGSE is how best to develop games that can exploit such capabilities, rather than supporting current/legacy games that were developed without dependence or integration of cloud-based services. For example, cloud-based CG R&D efforts are seeking to host game play computations in the cloud while streaming visual and audio content updates to remote player clients. This resembles a "fat server, thin client" architectural model that contrasts to the currently "thin server, fat client" model more common in networked multiplayer games. Both models, along with others like massively scaled peer-to-peer or Hypergrid architectures [15], represent different assumptions about preferred business models, financial investments, intellectual property protection, network bandwidth and latency quality of service, and anticipated advances in computing technology.

Cloud-based streaming of game content/sessions is also being actively pursued by large IT firms, perhaps inspired by the success of asynchronous, on-demand streaming video/audio distribution ventures that are displacing traditional synchronous, on-schedule media content broadcasters. But game play is interactive and experiential in ways that passive media (films, recorded music, television programs) are not. So it is not surprising to see that game play streaming services often provide reduced visual display resolution (to reduce bandwidth and latency) with game content that can be more easily cached (e.g., pre-recorded game video content), since wide-area network protocols are not yet developed to meet the performance and "quality of experience" (QoE) requirements that game players expect [17]. Thus CGSE challenges here include identifying how best to design games for cloud-based deployment infrastructure, and how to design the software infrastructure (including network protocols) for cloud-based games [13].

3.3 Game software repositories and data management services

The development, use, and ongoing support for CG requires or benefits from systematic approaches to game data management. As CG often support in-game objects, levels, or worlds as structured data sets that may span megabytes or gigabytes of data, that may often need to be rolled in/out depending on game play dynamics, then game data management emerges as a core technical challenge in game software engineering. At least five repositories or formal databases can be found supporting CGSE: (1) repositories for capturing game play telemetry data for analytics applications [33]; (2) game source code and content asset files and directories with version management (e.g., GitHub); (3) online game publisher repositories for remote database services for user registration, account management, player-specific in-game resource holdings and character personalizations, and (5) external game asset repositories and currency/banking services (micro-transactions supporting game asset purchase for free-to-play games).

Many challenges follow from different ways and means for orchestrating the movement of game data and resource files within and between repositories. This also points to challenges that arise when seeking to identify how best to manage game resources associated with game characters that can transition across game server shards, networked execution domains, or interoperable virtual worlds [15,24] via teleportation or similar methods. Game characters are increasingly becoming mobile code objects or databases, so how best to engineer game software data management across repositories will likely remain an active area of R&D interest, including within the commercial game industry.

4. DISCUSSION

The areas identified throughout this paper for future research in CGSE are a small sample rather than a comprehensive set. CGSE itself represents a new arena for SE research and development. Other CGSE challenges may be found in how best to:

* engineer cybersecurity software capabilities for multiplayer games where financial services, user privacy, and anti-cheating requirements must be addressed while insuring online socialization and convivial/competitive game play;

* integration of CG with social media services and other cloud-based services;

* developing frameworks or SDKs for implementing games that incorporate heterogeneous devices arising within an *Internet of Things*;

* develop automated or semi-automated game production tools that allow non-engineers to rapidly produce and deploy serious games for non-entertainment and scientific research applications [28];

* engineering remote game client services for display update management and run-time monitoring to improve game QoE;

* investigate software architectural alternatives for orthogonal, multi-tier middleware services (for game character AI, data analytics, financial payment services, user account management, etc.) developed by independent game service providers;

* design of rapid, iterative and incremental approaches to develop minimally playable games (or game feature sets) that allow

remote user play to help drive subsequent game play features and functionality, as well as how to adapt such an approach to non-game applications;

* articulating techniques that combine playtesting with play analytics and advanced visualizations (e.g., using synthetic, procedurally generated game worlds to visualize game play data sets and temporal relationships) [33];

* develop games, game engines, or game development frameworks for educational games in other CS sub-disciplines (e.g., serious games for network protocol design, database management, compiler construction, operating system configuration); and

* investigate the roles of FOSS tools and techniques for game development in enabling SE Education, global knowledge transfer, and socio-economic development of local game development enterprises.

The concept and the initiative of using games to explore new ways and means for solving long-standing problems in SE is profound, bold and ambitious. Whether it succeeds is an open issue for now. But whether it can and will be adopted to explore other challenges in SE is therefore an opportunity at hand. Similarly, whether such effort can be mobilized and deployed on a regional, national, or global scale without a large government research investment is also a challenge to be addressed. However, we should look for (and perhaps encourage) such efforts to address challenges in CGSE, whereby game software development serves game play whose purpose is to inform new ways to design games.

5. CONCLUSIONS

This paper reviews and identifies a number of outstanding R&D problems that can be found at the intersection of computer game and software engineering. Students and scholars of games or software development may therefore find a new fertile space for emerging research opportunity to explore, cultivate or embrace. The research objective in this paper is to share these results with others looking for challenging problems to pose and pursue.

Overall, the future of CGSE is filled with many diverse opportunities for research studies and technology development. These opportunities may be of some interest to the extant computer game industry, but the likelihood of interest for near-term research funds may be found outside of the entertainment-focused computer game studios [28]. Whether government agencies and corporate sponsors who do support high-risk research projects will embrace CGSE as a new opportunity area is an open question, but one that readers of this paper may be compelled to advocate and pursue.

Finally, we do not expect the world of SE to suddenly pivot around the emerging area of CGSE, though it might be interesting to see and experience such a transformation. Similarly, it will take time for our colleagues in mainstream Computer Science to come to recognize how computer games are engaging a new generation of students and researchers, though the emergence of new academic research centers and degree programs for (Computer) Game Science are just beginning to appear. Nonetheless, we believe there is a future to research in CGSE, and this future may need to find its home and advocates in the periphery of disciplines where new lines of research and practice are more likely to emerge. Thus, we encourage you to do so to pursue your interests in CGSE research and practice.

6. ACKNOWLEDGMENTS

Research support for Scacchi was provided by grant #1256593 from the National Science Foundation. No review, approval, or endorsement is implied. Materials appearing in Sections 2 and 3 also appear in [31], and are included by permission of the publisher.

7. REFERENCES

[1] Alspaugh.T.A. and Scacchi, W. (2013). Ongoing Software Development without Classical Requirements, *Proc. 21st IEEE Intern. Conf. Requirements Engineering* (RE'21), Rio de Janeiro, Brazil, 165-174, 15-19 July 2013.

[2] Bartle, R.A. (2004). *Designing Virtual Worlds*, New Riders, Indianapolis, IN.

[3] Bendifallah, S. and Scacchi, W. (1989). Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork, *Proc. 11th. Intern. Conf. Software Engineering*, Pittsburgh, PA, ACM and IEEE Computer Society, 260-270.

[4] Boehm, B., Gray, T., and Seewaldt, T. (1984). Prototyping Versus Specifying: A Multiproject Experiment. *IEEE Trans. Software Engineering*, 10(3): 290–303.

[5] 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.

[6] 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, Portugal.

[7] Cooper, K. and Longstreet, C. (2015) Integrating Learning Objectives for Subject Specific Topics and Transferable Skills, in Cooper, K. and Scacchi, W. (Eds.), *Computer Games and Software Engineering*, CRC Press, Taylor & Francis, Boca Raton, FL (to appear).

[8] Cooper, K., Nasr, E., and Longstreet, C.L. (2014). Towards Model-Driven Requirements Engineering for Serious Educational Games: Informal, Semi-formal, and Formal Models. In *Proc. 20th Intern. Working Conference on Requirements Engineering: Foundation for Software Quality*, Lecture Notes in Computer Science, V. 8396, 17-22.

[9] Dorling, A. and McCaffery, F. (2012). The Gamification of SPICE, in *Software Process Improvement and Capability Determination*, Communications in Computer and Information Science, Volume 290, Springer, 295-301.

[10] Elliott, M.S., Ackerman, M.S., and Scacchi, W. (2007). Knowledge Work Artifacts: Kernel Cousins for Free/Open Source Software Development, *Proc. ACM Conf. Support Group Work (Group07)*, Sanibel Island, FL, 177-186, November.

[11] Fernandes, J., Duarte, D., Ribeiro, C., *et al.* (2012). iThink: A Game-Based Approach Towards Improving Collaboration and Participation in Requirement Elicitation, *Procedia Computer Science*, 15, 66-77, DOI: 10.1016/j.procs.2012.10.059

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

[13] Huang, C-Y., Chen, K-T., Chen, D-Y, *et al.* (2014). Gaming Anywhere: The First Open Source Cloud Gaming System, *ACM Trans. Multimedia Computing Communications and Applications*, 10(1).

[14] Logas, H., Whitehead, J., Mateas, M., *et al.* (2014), Software Verification Games: Designing Xylem, The Code of Plants, *Proc. 9th International Conf. Foundations of Digital Games (FDG 2014)*, Ft. Lauderdale, FL, USA.

[15] Lopes, C. (2011). Hypergrid: Architecture and Protocol for Virtual World Interoperability, *IEEE Internet Computing*, 15(5), 22-29.

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

[17] Mishra, D., El Zarki, M., Erbad, A., Hsu, C-H., and Venekatasubramanian, N. (2014). Clouds + Games: A Multifacted Approach, *IEEE Internet Computing*, May-June 2014, 20-27.

[18] Murphy-Hill, E., Zimmerman, T., and Nagappan, N. (2014). Cowboys, Ankle Sprains, and Keepers of Quality: How is Video Game Development Different from Software Development? *Proc. 36th Intern. Conf. Software Engineering* (ICSE 2014), ACM, Hyderabad, India, 1-11, June.

[19] Musil, J. Schweda, A., Winkler, D., and Biffl, S. (2010). Synthesized Essence: What Game Jams Teach about Prototyping of New Software Products, *Proc. 32nd Intern. Conf. Software Engineering* (ICSE'10), ACM, Cape Town, SA, 183–186.

[20] Preston, J. A., Chastine, J., O'Donnell, C., Tseng, T., & MacIntyre, B. (2012). Game Jams: Community, motivations, and learning among jammers. *Intern. J. Game-Based Learning*, 2(3), 51-70.

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

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

[23] Scacchi, W. (2010). Collaboration Practices and Affordances in Free/Open Source Software Development, in I. Mistrík, J. Grundy, A. van der Hoek, and J. Whitehead, (Eds.), *Collaborative Software Engineering*, Springer, New York, 307-328, 2010.

[24] Scacchi, W. (2010). Game-Based Virtual Worlds as Decentralized Virtual Activity Systems, in W.S. Bainbridge (Ed.), *Convergence of the Real and the Virtual*, 225-236, Springer, New York.

[25] Scacchi, W. (2010). Computer Game Mods, Modders, Modding, and the Mod Scene, *First Monday*, 15(5), May.

[26] Scacchi, W. (2011). Modding as an Open Source Software Approach to Extending Computer Game Systems, *Intern. J. Open Source Software and Processes*, 3(3), 36-47, July-September 2011.

[27] Scacchi, W. (2012). Competitive Game Development: Software Engineering as a Team Sport. Keynote Address, *2nd Intern. Workshop on Games and Software Engineering* (GAS2012), Intern. Conf. Software Engineering, Zurich, CH., May 2012.

[28] Scacchi, W. (Ed.), (2012). *The Future of Research in Computer Games and Virtual Worlds: Workshop Report*, Technical Report UCI-ISR-12-8, Institute for Software Research, University of California, Irvine, Irvine, CA. July.

[29] Scacchi, W. (2015). Repurposing Game Mechanics as a Technique for Developing Game-Based Virtual Worlds ,in Cooper, K. and Scacchi, W. (Eds.), *Computer Games and Software Engineering*, CRC Press, Taylor & Francis, Boca Raton, FL (to appear).

[30] Scacchi, W. and Alspaugh, T.A. (2012). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *J. Systems and Software*, 85(7), 1479-1494, July.

[31] Scacchi, W. and Cooper, K.M. (2015). Emerging Research Challenges in Computer Games and Software Engineering, in Cooper, K. and Scacchi, W. (Eds.), *Computer Games and Software Engineering*, CRC Press, Taylor & Francis, Boca Raton, FL (to appear).

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

[33] Seif El-Nasr, M., Drachen, A.,Canossa, A. (2013). *Game Analytics: Maximizing the Value of Player Data*, Springer, New York.

[34] Sheldon, L. (2011). *The Multiplayer Classroom: Designing Coursework as a Game*, Cengage Learning PTR, Independence, KY.

[35] Shin, K., Kaneko, K., Matsui, M., *et al.* (2012). Localizing *Global Game Jam*: Designing Game Development for Collaborative Learning in the Social Context, in Nijholt, A. Romano, T., and Reidsma, D., (Eds). *Advances in Computer Entertainment*, Lecture Notes in Computer Science, Vol. 7624, Springer, Berlin, 117-132.

[36] Smelik, R.M., Tutenel,T., Bidarra, R., and Benes, B. (2014). A Survey on Procedural Modeling for Virtual Worlds, *Computer Graphics Forum*, DOI:10.1111/cgf.12276

[37] Smite, D. and Wohlin, C. (2011). A Whisper of Evidence in Global Software Engineering, *IEEE Software*, 28(4), 15-18, July-August.

[38] Smith, G., Whitehead, J., Mateas, M., et al. (2011). Launchpad: A Rhythm-Based Level Generator for 2-D Platformers, *IEEE Trans. Computational Intelligence and AI in Games (TCIAIG)*, 3(1), March 2011.

[39] VERIGAMES (2015). http://www.verigames.com/, accessed June 2014 and February 2015.

[40] Wang, A.I. (2015). The Use of Game Development in Computer Science and Software Engineering Education, in Cooper, K. and Scacchi, W. (Eds.), *Computer Games and Software Engineering*, CRC Press, Taylor & Francis, Boca Raton, FL (to appear).

[41] Xie, T., Tillman, N., de Halleux, J. and Bishop, J. (2015). Educational Software Engineering: Where Software Engineering, Education, and Gaming Meet, in Cooper, K. and Scacchi, W., *Computer Games and Software Engineering*, CRC Press, Taylor & Francis Inc., Baco Raton, FL (to appear).

[42] Zook, A. and Riedl. M.O. (2013). Game Conceptualization and Development Processes in the Global Game Jam, *Proc. Foundations of Digital Games Workshop on the Global Game Jam*, Crete, GR.