The Whats and the Whys of Games and Software Engineering

Chris Lewis, Jim Whitehead
University of California, Santa Cruz
1156 High St, Santa Cruz, California, USA
{cflewis,ejw}@soe.ucsc.edu

ABSTRACT

The intersection of video games and software engineering is not yet well understood. This paper highlights the varied and exciting opportunities available at the intersection of these two disciplines. We investigate four main areas: the development of games, how they are designed, how middleware supports the creative process and how games are tested. We hope that it inspires readers to take on the challenges available in games and software engineering, and join together to create a vibrant community.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.2 [**Software Engineering**]: Testing and Debugging; D.2.9 [**Software Engineering**]: Management; K.8.0 [**Personal Computing**]: Games

General Terms

Design, Management

Keywords

video games, software engineering, survey

1. INTRODUCTION

When traveling to academic conferences, we are often asked: "Why are video games important? Why should I care?" The mere suggestion that games require special treatment often challenges those who believe that software engineering, regardless of domain, is fundamentally the same. When speaking to game developers, we hear the flip side of the coin: "Why is software engineering important? If I needed this, I would be using it already. This looks like a waste of time." As one can tell, working at the intersection of games and software engineering can be disheartening.

Games and software engineering have important things they can learn from one another. Games contain a confluence of interesting properties, such as emergence, real-time interaction and computationally-challenging components that come together to create a new field of study, despite its familiar parts. Software engineering has much to help aid the infant games industry, an industry that has largely discounted software engineering as being unable to meet its needs. To aid the problems the games industry has, solutions will have to satisfy tight computational constraints. We believe these solutions may well be able to be used in many other domains where the restrictions are looser.

In this paper, we describe research areas that present themselves between the two worlds of software engineering and game development. They represent rich seams of intellectual investigation, while providing direct, provable benefits to software development at large.

GAS '11, May 22, 2011, Waikiki, Honolulu, HI, USA

2. RESEARCH AREAS

2.1 Overview

Identifying valuable areas of research for games can be troublesome. The games industry largely operates as a black-box, full of proprietary code and non-disclosure agreements that can make it difficult for outsiders to try to navigate.

To aid us in this task, we use two main papers. The first, by independent game developer Jonathan Blow, appeared in the *ACM Queue* magazine in 2004 [3]. In it, he describes what he considers problems related to project size and complexity, and those related to "highly domain-specific" requirements. This landmark paper provides unique insight into the challenges posed by modern game development. The second paper we use is by Tschang, who studied and categorized 65 different game development post-mortems reported in *Game Developer* magazine [28]. These post-mortems provide a valuable insight into the development process, listing five things that go right and that go wrong, that have been described as "fascinating war stories of extremely challenging software development projects" [1].

To answer the perennial "Why should software engineers care about games?" question, we look into four main areas where we believe software engineering and games have much to learn from each other: development, design, middleware and testing. We'll look into *what* the problems in games are, and *why* they are of interest to the software engineering community at large.

2.2 Development

2.2.1 Process

What In Tschang's review of postmortems, the top two most cited failures in a development project are either pipeline or team related. The game development pipeline specifies each stage of the project, typically what its dependencies are and estimates of how long each stage will take. Optimal pipeline development is often difficult for game teams as they are both multifunctional and tightly coupled. For example, the art team cannot reasonably begin work before the specifications of the game engine are known. However, tight budgets and deadlines means studios cannot afford for employees to be under-utilized.

Video games also suffer from more traditional human factors: Tschang notes that larger team sizes require strong leadership. We have found in personal interviews examples of lack of documentation preventing reasonable knowledge dissemination and encouraging code rot of valuable assets, and these problems are exacerbated by employee turnover. Agile development is beginning to gain traction with game developers, particularly SCRUM.

Many articles and books have been written discussing such issues [4, 8, 15, 26].

Why The multifunctional teams that game developers operate are not unique to the computing industry. Apple, for example, is often praised for its marriage of functionality and design, which is only achievable by incorporating employees well-versed in computer science, human-computer interfaces and industrial design. The intense constraints that game development finds itself under, financially, time-wise and technically, means game development is a challenging software engineering area. Gaining insight into how these teams operate, and how they can operate better, will undoubtedly benefit software development as a whole.

2.2.2 Tools and Environments

What As Blow puts it, "To tackle such complexity, it helps to have excellent development tools. Sadly, we do not have excellent development tools." Blow laments that current tools, like Visual Studio are developed with a general audience in mind, and development effort spent on unused features is effort wasted for the games industry, leading to domain-specific tools like the UnrealEd level editor for Unreal Engine games. Since the article's publication, specialized toolsets, such as Microsoft's XNA, have appeared, aiding the game development process.

We were unable to find any survey of developers that looks into what development requirements they have, how they are supporting themselves in these requirements, and how they would like their requirements to be met in an ideal world. Such a survey would be highly beneficial in targeting future research.

Why The games industry typically works at the bleeding edge of technical complexity. Developing tools and environments to help support their efforts provides interesting, deep problems that will lead to the creation of better tooling and provide insight into creating domain-specific environments.

2.3 Design

2.3.1 Design Patterns

What Design patterns, such as those from the Gang of Four [11], have proven to be an invaluable means of communicating solutions to common problems. However, game designers have had difficulty finding a similar language with which to speak. Broad genres can be applied to games at a high-level, but the building blocks of design remain vaguely described, dooming developers to repeating the same problems over and over. Church identifies this lack of a domain-specific language: "We should be able to play a side-scrolling shooter on a Game Boy, figure out one cool aspect of it, and apply that idea to the 3D simulation we're building... If we reach this level of understanding, evolution of design across all genres will accelerate... we need a shared language of game design" [6].

Examples of design pattern research into games include [2, 13, 16].

Why A greater understanding of design patterns benefits Computer Science, but in particular, broadens our understanding of the technique as software engineers. While the Gang of Four proved the value of patterns to object-oriented programming, and patterns are used successfully in UI problems, further expansion of the technique may reveal applicability to more domains. Could successful software engineering simply employ patterns from design, to code, to user experience?

2.3.2 Formal Models

What Formal models are rigorously defined, unambiguous specifications. In computer science, formal models are usually used in order to leverage computational verification support, such as model checking.

Many game designs are succinct in nature, and hinge on simply expressed rules, making them prime candidates for modeling. Using formal models allows the designer to explore their game design, using both manual and automated techniques to simulate playthroughs and verify properties. This modeling process is a cheaper and faster method of gaining design insight than the creation of a prototype, and provides the opportunity to refine the design before any play testing has occurred.

Examples include Machinations [7] and Ludocore [24].

Why Games, while being easily expressed as a model, invariably suffer from state space explosion. Model checking, the primary automatic model verification technique, has to build the state space first; an intractable proposition for all but the simplest games. New methodologies for working with formal models are required, and these methodologies will allow other software domains to check more complicated models. Additionally, many find formal modeling constrictive and terse.

The loosening of requirements for checking games may result in frameworks or methodologies that more easily fit with many more software engineers.

2.3.3 Creativity Support

What Creativity support is a field that straddles software engineering, human-computer interfaces, and artificial intelligence. The goal is to develop a workflow where humans and computers work in tandem, with the human's design being aided and evaluated by the computer. This is an extension of Computer-Aided Design, where the computer largely takes a passive role. Negroponte referred to such creativity support tools as "design amplifiers" [20].

Games are particularly amenable to creativity support as it is often difficult for a human to maintain an accurate mental model of all the affordances and constraints that a game design offers, whereas a computer is able to quickly assess how modifications to a game's rules and level design may affect other components. While there have been calls for creativity support for games [14, 23], only few have heeded it, such as Smith's tool for platformer level design [25].

Why Software engineering, as we know it, places the computer in a largely passive position. Most of the CPU cycles available while engineers are developing go unused; cycles that could be put to use to help aid the creation of better software. It is likely the future of software engineering will more heavily involve artificial intelligence techniques to utilize this resource, and investigations into creativity support will provide a necessary cornerstone for this work.

2.4 Middleware

What The technical challenges of game design provides a significant barrier to the actual creation of an enjoyable game. A burgeoning middleware industry has grown up to meet this need, with everything from general purpose tools such as graphics engines (Unreal Engine, CryEngine, Source) and physics engines (Havok), all the way to entirely specialized tools such as foliage generators (SpeedTree), pathfinding algorithms (PathEngine) and multiplayer support (GameSpy) [10].

However, a significant systems integration problem has now arisen, with Tschang noting that engines often need rewriting to provide specialized features. Additionally, support agreements can now affect development, famously resulting in a legal tussle between Silicon Knights and Epic Games, with Silicon Knights contending Epic Games were unable to deliver their engine on time as promised [5].

Why Surveys into the skills required to successfully integrate middleware have yet to be performed, and one can imagine a full-time role for a systems integration engineer as part of a game development team. What are the strengths and weaknesses of such an approach?

2.5 Testing

2.5.1 Game Testing

What Emergent software is a specific class of software where the system outputs are unpredictable. Usually, emergence is created by multiple simple subsystems interacting in such a manner to create unpredictable results. Almost all games are emergent; their ability to surprise players is key to their long-term appeal [21]. However, emergent software is resistant to automated testing techniques, which require upfront knowledge of expected output. Additionally, designers want to ask questions like "Do 90% of players finish the game in less than 5 minutes?" not unit testable requirements, like "Does function $squareValue(int)$ return 4 when supplied with 2?" Such questions pose traditional usability testing problems.

The current state of the art in games testing is to hire hundreds of human testers to manually play builds of the game at various stages of the development process [27].

The only active research into testing emergent software, that we are aware of, is [17].

Why Software across all domains is beginning to incorporate more technologies that will lead to emergence, particularly artificial intelligence. Engineers will be less and less able to control and predict the outputs of their systems, and their ability to validate and verify their software will greatly diminish. Research into emergence now will provide all developers with the tools they require to verify their software in the future.

2.5.2 GUI Testing

What Almost all games are interacted with via a GUI. The user interface provides the gamer with a view into the world, and thus is one of the most important aspects of games. This means, however, that games suffer from the same GUI testing difficulties as other domains.

GUI testing is still in its infancy. [19] provides a useful summation of the challenges.

Why Games may provide a good first step for GUI testing. The GUIs in many games are generally minimalist; providing the player with just enough information as necessary, and limiting the player's interaction with various configuration screens. Effective GUI testing may be more tractable in this space than traditional desktop software. Interestingly, this problem may also benefit from an artificial intelligence solution. AI techniques could be used to create agents, both in and out of the game world, that exercise game GUIs. These may translate to other, non-game domains.

3. THE GAMES DIFFERENCE

The one unique aspect of games, that seems to separate it from traditional software development, is the requirement for games to be *fun*. This requirement, unlike many others that we find in software engineering, has no metric that

can be applied; it is purely subjective. Indeed, what is fun for one audience may not be for another. However, fun must pervade the product and be supported by and validated at each stage of the development process. For this reason, games must be developed in a highly iterative manner [18]. Salen & Zimmermann describe how prototyping must begin as early as possible in order to begin the evaluation of a game design [22]. They note that digital game designers have tried to design a game upfront through copious amounts of documentation, but that the documentation is made instantly obsolete by surprises that arise when actually implementing the game. A classic example of this is the late 90's game *Jurassic Park: Trespasser*, whose specification focused gameplay on physics-based puzzles. The physics engine implementation was unable to support these puzzles, which forced a late-project shift to shooter-based gameplay [12]. Constant experimentation is key to a successful game creation effort, and all aspects of development, from toolsets to workflows to testing have to support this development style.

However, with this said, fun does have a strong relationship with software development companies that focus on *user experience*. Again, we are reminded of Apple, who have been described as being at the forefront of "how computing became fun" [9]. Fun might not be a core part of all software, but it certainly seems to make user-facing software more successful. This indicates that borrowing practices from game development and using them in user-facing software development could be beneficial.

4. CONCLUSION

In this paper, we've highlighted some of the completely unexplored research areas available, which will not only benefit games, but software engineering as a whole. Games are a domain with such unique constraints that they force scholars and practitioners into finding new ways of looking at problems. Often, these new insights provide us with a wider understanding of a particular problem. Sometimes, a new issue is presented, one that will undoubtedly affect other domains in time. And perhaps, new insights will provide a more elegant, adaptable and generalized view of an existing issue, immediately benefiting software engineering as a whole.

5. REFERENCES

[1] Atwood, J. Trespasser Postmortem, December 2005. Available from World Wide Web: http://bit.ly/atwood-trespasser.

[2] Bjork, S., and Holopainen, J. *Patterns in Game Design (Game Development Series)*, 1 ed. Charles River Media, December 2004.

[3] Blow, J. Game Development: Harder Than You Think. *Queue 1*, 10 (2004), 28–37.

[4] Campbell, B. Swiss Army Chainsaw: A Common Sense Approach to Tool Development. *Gamasutra* (September 2006).

[5] Carless, S. Breaking: Silicon Knights Files Lawsuit Against Epic. *Gamasutra* (July 2007).

[6] Church, D. Formal Abstract Design Tools. *Gamasutra* (July 1999).

[7] Dormans, J. Machinations: Elemental Feedback Patterns for Game Design. In *GAME-ON-NA 2009: 5th International North American Conference on Intelligent Games and Simulation* (2009), pp. 33–40.

[8] Fisch, I. 10 Game Design Process Pitfalls. *Gamasutra* (May 2009).

[9] Fry, S. The iPad launch: Can Steve Jobs do it again? *Time* (Apr. 2010).

[10] Funge, J. Let Us Entertain You. *Computer 41*, 12 (2008), 120–122.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, November 1994.

[12] Grossman, A. *Postmortems from Game Developer*. Focal Press, March 2003.

[13] Hullett, K., and Whitehead, J. Design patterns in FPS levels. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG '10)* (2010), pp. 78–85.

[14] Isla, D. Next-Gen Content Creation for Next-Gen AI. Speech, Foundations of Digital Games 2009, April 2009.

[15] Keith, C. *Agile Game Development with Scrum*. Addison-Wesley Professional, June 2010.

[16] Kreimeier, B. The Case For Game Design Patterns. *Gamasutra* (March 2002).

[17] Lewis, C., and Whitehead, J. Runtime repair of software faults using event-driven monitoring. In *ICSE '10: Proceedings of the 32nd International Conference on Software Engineering* (2010), pp. 275–280.

[18] Luton, W. Making Better Games Through Iteration. *Gamasutra* (October 2009).

[19] Memon, A. M. GUI testing: pitfalls and process. *Computer 35*, 8 (August 2002), 87–88.

[20] Negroponte, N. *Soft Architecture Machines*. The MIT Press, February 1976.

[21] Salen, K., and Zimmerman, E. *Rules of Play*. MIT Press, 2004, pp. 350–353.

[22] Salen, K., and Zimmerman, E. *Rules of Play*. MIT Press, 2004, p. 12.

[23] Satchell, C. Evolution of the Medium – Positioning for the Future of Gaming. Speech, Foundations of Digital Games 2009, April 2009.

[24] Smith, A. M., Nelson, M. J., and Mateas, M. Ludocore: A Logical Game Engine for Modeling Videogames. In *Conference on Computational Intelligence and Games* (August 2010).

[25] Smith, G., Whitehead, J., and Mateas, M. Tanagra: a mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG '10)* (2010), pp. 209–216.

[26] Spaulding, S. *Team Leadership in the Game Industry*. Course Technology PTR, January 2009.

[27] Starr, K. Testing Video Games Can't Possibly Be Harder Than an Afternoon With Xbox, Right? *Seattle Weekly* (July 2007).

[28] Tschang, F. T. Videogames As Interactive Experiential Products And Their Manner Of Development. *International Journal of Innovation Management 9*, 01 (2005), 103–131.