Modding as a Basis for Developing Game Systems

Walt Scacchi
Institute for Software Research
University of California, Irvine
wscacchi@ics.uci.edu

ABSTRACT

This paper seeks to briefly examine what is known so far about game mods and modding practices. Game modding has become a leading method for developing games by customizing extensions to game software. The research method in this study is observational and qualitative, so as to highlight current practices and issues that can be associated with software engineering foundations. Numerous examples of different game mods and modding practices are identified throughout.

Categories and Subject Descriptors

D.2 [Software Engineering]: s*oftware development, software architecture, design methodology*

General Terms

Design, Human Factors.

Keywords

Computer games, software extension, game modding

1. INTRODUCTION

User modified computer games, hereafter *game mods*, are a leading form of user-led innovation in game design and game play experience. But modded games are not standalone systems, as they require the user to have an originally acquired or authorized copy of the unmodded game.

Modding, the practice and process of developing game mods, is typically a “Do It Yourself” (DIY) approach to end-user game software engineering [2] that can establish both social and technical knowledge for how to innovate by resting control over game design from their original developers. At least four types of game mods can be observed: user interface customization; game conversions; machinima; and hacking closed game systems. Each enables different kinds of extension to the base game or game run-time environment. Game modding tools and support environments that support the creation of such extensions also merit attention. Subsequently, we conceive of game mods as covering customizations, tailorings, and remixes—that is, *software extensions* —of game embodiments, whether in the form of game content, software, or hardware denoting our space of interest.

The most direct way to become a game modder is through self-tutoring and self-organizing practices. Modding is a form of learning – learning how to mod, learning to be a game developer, learning to become a game content/software developer, learning computer game science outside or inside an academic setting, and more [3,13]. Modding is also a practice for learning how to work with others, especially on large, complex games/mods. Mod team efforts may also self organize around emergent software development project leaders or “want to be” (W.T.B.) leaders, as seen for example in the *Planeshift* open source MMOG development/modding project [13].

GAS'11, May, 2011, Honolulu, HI, USA.
Copyright 2011 ACM 1-58113-000-0/00/0010…$10.00.

Game mods, modding practices, and modders are in many ways quite similar to their counterparts in the world of free/open source software development (FOSSD). Modding is to games, like FOSSD is to software — they are increasingly becoming a part of mainstream technology development culture and practice. Modders are players of the games they construct, just like FOSS developers are also users of the systems they develop. There is no systematic distinction between developers and users in these communities, other than there are users/players that may contribute little beyond their usage, word of mouth they share with others, and their demand for more such systems. At FOSSD portals like SourceForge.com, as of January 2011, indicates the domain of “games” appears as the third most popular project category with over 23K active projects. These projects develop either FOSS-based games, game engines, or game tools/SDKs, and all of the top 50 projects each have logged more than 1M downloads. So the intersection of games and FOSS covers a substantial social and technological plane, as both modding and FOSS development are participatory, user-led modes of system development that rely on continual replenishment of new participants joining and migrating through project efforts, as well as new additions or modifications of content, functionality and end-user experience [12,13,14]. Modding and FOSSD projects are in many ways experiments to prototype alternative visions of what innovative systems might be in the near future, and so both are widely embraced and practiced primarily as a means for learning about new technologies, new system capabilities, new working relationships with potentially unfamiliar teammates from other cultures, and more [cf. 14].

Consequently, game modding can be recognized as a leading method for developing or customizing game software. And software extensions are the leading ways and means for game modding.

This paper seeks to briefly examine what is known so far about game mods and modding practices. The research method in this study is observational and qualitative, so as to highlight current practices and issues that can be associated with software engineering foundations. Numerous examples of different game mods and modding practices are identified throughout to help distinguish empirically grounded observation from conjecture. All of the types of game mods and modding practices identified in this paper have been employed first-hand by game development projects led or produced by the author. Such observation can subsequently serve as a basis for further empirical study and technology development that ties together computer games and software engineering [12,13,14].

2. SOFTWARE EXTENSION

Game mods embody different techniques and mechanisms for software extension. However, the description of game mods and modding is often absent of its logical roots or connections back to software engineering. As suggested, mods are extensions to existing game software systems, so it is appropriate to review what we already know about software extensions and extensibility.

Parnas [10] provides an early notion of software extension as an expression of modular software design. Accordingly, modular systems are those whose components can be added, removed, or updated while satisfying the original system functional requirements. Such concepts in turn were integrated into software architectural design language descriptions and configuration management tools [7]. But reliance on software architecture descriptions is not readily found in either conventional game or mod development. Hentonnen and colleagues [6] examine how software plug-ins support architectural extension, while Leveque, et al. [8] investigate how extension mechanisms like views and model-based systems support extension also at the architectural level. Last, the modern Web architecture is itself designed according to principles of extensibility through open interfaces, migration across software versions, network data content/hypertext transfer protocols, and representational state transfer [4]. Mod-friendly networked multi-player games appear to take advantage of these capabilities.

In contrast, Batory and associates [1] describe how domain-specific (scripting) languages and software product lines provide support software extension, and it now seems clear that such techniques are commonly used in games that are open for modding. Finally, FOSSD has become another approach to extensible software engineering in practice [14]. So software extensions and extensibility is a foundational concept in software engineering, and thus to no surprise, also foundational to the development of game mods. However, the logical connections and common/uncommon legacy remain under specified, which this paper seeks to address and update.

3. FOUR TYPES OF GAME MODS

3.1 User interface customizations

User interfaces to games embody the practice and experience of interfacing users (game players) to the game system and play experience designed by game developers. Game developers act to constrain and govern what users can do, and what kinds of experience they can realize. Some users in turn seek to achieve some form of competitive advantage during game play by modding the user interface software for their game, when so enabled by game developers, to acquire or reveal additional information that the users believe will help their play performance and experience. User interface add-ons subsequently act as the medium through which game development studios support game product customization as a strategy for increasing the likelihood of product success through end-user satisfaction [2].

Three kinds of user interface customizations can be observed. First and most common, is the player's ability to select, attire or accessorize a *player's in-game identity*. Second, is for players to customize *the color palette and representational framing borders* of the their game display within the human-computer interface, much like what can also be done with Web browsers and other end-user software applications. Third, are *user interface add-on modules* that modify the player's in-game information management dashboard that do not modify game play rules or functions. These add-ons provide additional information about game play or game state that may enhance the game play experience, as well as increasing a player's sense of immersion or omniscience within the game world through sensory or perceptual expansion. This in turn enables awareness of game events not visible in the player's current in-game view. Consequently, the first two kinds of customizations result from meta-data selections within parametric system functions, while the third represents a traditional kind of modular extension that does not affect the pre-existing game's functional requirements.

3.2 Game conversions

Game conversion mods are perhaps the most common form of game mods. Most such conversions are partial, in that they add or modify (a) in-game characters including user-controlled character appearance or capabilities, opponent bots, cheat bots, and non-player characters, (b) play objects like weapons, potions, spells, and other resources, (c) play levels, zones, maps, terrains, or landscapes, (d) game rules, or (e) play mechanics. Some more ambitious modders go as far as to accomplish (f) total conversions that create entirely new games from existing games of a kind that are not easily determined from the originating game. For example, one of the most widely distributed and played total game conversions is the *Counter-Strike* (CS) mod of the *Half-Life* (HL) first-person action game from Valve Software. As the success of the CS mod gave rise to millions of players preferring to play the mod over the original HL game, then other modders began to access the CS mod to further convert in part or full, to the point that Valve Software modified its game development and distribution business model to embrace game modding as part of the game play experience that is available to players who acquire a licensed copy of the HL product family. Valve has since marketed a number of CS variants that have sold over 10M copies as of 2008, thus denoting the most successful game conversion mod, as well as the most lucrative in terms of subsequent retail sales derived from a game mod.

Another example is found in games converted to serve a purpose other than entertainment, such as the development and use of games for science, technology, and engineering applications. For instance, the *FabLab* game [15] is a conversion of the *Unreal Tournament 2007* retail game, from a first-person action shooter to a simulator for training semiconductor manufacturing technicians in diagnosing and treating potentially hazardous materials spills in a cleanroom environment. However, this conversion is not readily anticipated by knowledge of the Unreal games or underlying game engine, though it maintains operational compatibility with the Unreal game itself. So game conversions can repurpose the look, feel, and intent of a game across application domains, while maintaining a common software product line [cf. 1].

Finally, it is common practice that the underlying game engine has one set of license terms and conditions to protect original work (e.g., no redistribution), while game mod can have a different set of terms and conditions as a derived work (e.g., redistribution allowed only for a game mod, but not for sale). In this regard, software licenses embody the business model that the game development studio or publisher seeks to embrace, rather than just a set of property rights and constraints. For example, in *Aion*, an MMOG from South Korean game studio NCSoft, no user created mods or user interface add-ons are allowed. Attempting to incorporate such changes would therefore conflict with its EULA and subsequently put such user-modders at risk of losing their access to networked *Aion* multi-player game play. In contrast, the MMOG *World of Warcraft* allows for UI customization mods and add-ons only, but no other game conversions, no reverse engineering game engine, and no activity intended to bypass WoW's encryption mechanisms. And, in one more variation, for games like *Unreal Tournament*, *Half-Life*, *NeverWinterNights*, *Civilization* and many others, the EULAs encourage modding and the free redistribution of mods without fee to others who must have a licensed game copy, but no reverse engineering or redistribution of the game engine required to run the mods. This restriction in turns helps game companies realize the benefit of increased game sales by players who want to play with known mods, rather than with the unmodded game as sold at retail. Mods thus help improve games software sales, revenue, and profits for the game development studio, publisher, and retailer

3.3 Machinima

Machinima can be viewed as the product of modding efforts that intend to modify the visual replay of game usage sessions. Machinima employ computer games as their creative media, such that these new media are mobilized for some other purpose (e.g., creating online cinema or interactive art exhibition). Machinima focuses attention to playing and replaying a game for the purpose of story telling, movie making, or retelling of daunting or high efficiency game play/usage experience [9]. Machinima is a form of modding the experience of playing a specific game through a recording of its visual play session history so as to achieve some other ends beyond the enjoyment (or frustration) of game play. These play-session histories can then be further modded via video editing or remixing with other media (e.g., audio recordings) to better enable cinematic storytelling or creative performance documentation. Machinima is thus a kind of play/usage history process re-enactment [cf. 11] whose purpose may be documentary (replaying what the player saw or experienced during a play session) or cinematic (creatively steering a play session so as to manifest observable play process enactments that can be edited and remixed off-line to visually tell a story). Machinima mods are thus a kind of extension that is not bound to the architecture of the underlying game system, except for how the game facilitates a user's ability to structure and manipulate emergent game play to realize a desired play process enactment history.

3.4 Hacking closed game systems

Hacking a closed game system is a practice whose purpose oftentimes seems to be in direct challenge to the authority of commercial game developers that represent large, global corporate interests. Hacking proprietary game software is often focused not so much on how to improve competitive advantage in multi-player game play, but instead is focused on expanding the range of experiences that users may encounter through use of alternative technologies [5,13]. For example, Huang's [5] study instructs readers in the practice of "reverse engineering" as a strategy to understand both how a game platform was designed and how it operates in fine detail, as a basis for developing new innovative modifications or original platform designs, such as installing and running a FOSS-based Linux operating system (instead of Microsoft's proprietary closed source offering). While many game developers seek to protect their intellectual property (IP) from reverse engineering through end-user license agreements whose terms attempt to prohibit such action under threat of legal action, reverse engineering is not legally prohibited nor discouraged by the Courts. Consequently, the practice of modding closed game systems is often less focused on enabling players to achieve competitive advantage when playing retail computer games, but instead may encourage those few so inclined for how to understand and ultimately create computing innovations through reverse engineering or other DIY game system modifications. Closed game system modding is thus a style of software extension by game modders who are willing to fore go the "protections" and quality assurances that closed game system developers provide, in order to experience the liberty, skill and knowledge acquisition, as well as potential to innovate, that mastery of reverse engineering affords. Consequently, players/modders who are willing to take responsibility for their actions (and not seek to defraud game developers or publishers due to false product failure warranty claims or copyright infringement), can enjoy the freedom to learn how their gaming systems work in intimate detail and to potentially learn about game system innovation through discovery and reinvention with the support of others like-minded [cf. 13].

Finally, games are one of the most commonly modified types of software that are transformed into "pirated games" that are "illegally downloaded." Such game modding practice is focused on engaging a kind of meta-game that involves modding game IP from closed to (more) open. Game piracy has thus become recognized as a collective, decentralized and placeless endeavor (i.e., not a physical organization) that relies on torrent servers as its underground distribution venue for pirated game software. As recent surveys of torrent-based downloads reveals, in 2008 the top 10 pirated games represented about 9M downloads, while in 2009 the top 5 pirated games represent more than 13M downloads, and in 2010 the top 5 pirated games approached 20M, all suggesting a substantial growth in interest in and access to such modded game products. Thus, we should not be surprised by the recent efforts of game system hackers that continue to demonstrate the vulnerabilities of different hardware and software-based techniques to encrypt and secure closed game systems from would be hackers. However, it is also very instructive to learn from these exploits how difficult it is to engineer truly secure software systems, whether such systems are games or some other type of application or package.

4. GAME MODDING SOFTWARE TOOLS AND SUPPORT

Games are most often modded with tools that provide access to an unencrypted representation of the game software or game platform. Such a representation is accessed and extended via a domain-specific (scripting) language. While it might seem the case that game vendors would seek to discourage users from acquiring such tools, we observe a widespread contrary pattern.

Game system developers are increasingly offering software tools for modifying the games they create or distribute, as a way to increase game sales and market share. Game/domain-specific Software Development Kits (SDKs) provided to users by game development studios represent a contemporary business strategy for engaging users to help lead product innovation from outside the studio. Once id Software, maker of the *DOOM* and *Quake* game software product line, and also Epic Games, maker of the *Unreal* software game product line, started to provide prospective game players/modders with software tools that would allow them to edit game content, play mechanics, rules, or other functionality, other competing game development studios were pressured to make similar offerings or face a possible competitive disadvantage in the marketplace. However, these tools do not provide access to the underlying source code that embodies the proprietary game engine —a large software program infrastructure that coordinates computer graphics, user interface controls, networking, game audio, access to middleware libraries for game physics, and so forth. But the complexity and capabilities of such a tool suite mean that any one, or better said, any game development or modding team, can now access modding tools or SDKs to build commercial quality games. But mastering these tools appears to be an undertaking likely to be only of interest to highly committed game developers who are self-supported or self-organized.

In contrast to game modding platforms provided by game development studios, there are also alternatives provided by the end-user community. One approach can be seen with facilities provided in *Garry's Mod* mod-making package that you can use to construct a variety of fanciful contraptions as user created art works, or to create comic books, program game conversions, and produce other kinds of user created content. But this package requires that you own a licensed game like *Counter-Strike: Source*, *Half-Life2* or *Day of Defeat: Source* from Valve Software.

A different approach to end-user game development platforms can be found arising from FOSS games and game engines. The *DOOM* and *Quake* games and game engines were released as free software subject to the GPL, once they were seen by id Software as having reached the end of their retail product cycle. Hundreds of games/engines have been developed and released for download starting from the FOSS that was the platform of the original games. However, the content assets for many of these games (e.g., in-game artwork) are not covered by the GPL, and so user-developers must still acquire a licensed copy of the original game if its content is to be reused in some way. Nonetheless, some variants of the user-created GPL'd games now feature their own content that is limited/protected by Creative Commons licenses.

5. OPPORTUNITIES FOR MODDING AND SOFTWARE ENGINEERING

Game modding demonstrates the practical value of software extension as a user-friendly approach to custom software. Such software can extend games open to modding into diverse product lines that flourish through reliance on domain-specific game scripting languages, and integrated software development kits. Modding also demonstrates the success of end-users learning how to extend software to create custom user interface add-ons, system conversions, replayable system usage documentaries and movies, as well as to discover security vulnerabilities. Game modding therefore represents a viable form of end-user engineering of complex software that may be transferable to other domains.

6. ACKNOWLEDGMENTS

The research described in this paper has been supported by NSF grants #0808783 and #1041918, grants #N00244-10-1-0064 and #N00244-10-1-0077 from the Naval Postgraduate School. No review, approval or endorsement implied.

7. REFERENCES

[1] Batory., D., Johnson, C., MacDonald, B., von Heeder, D., Achieving extensibility through product lines and domain specific languages: a case study, *ACM Trans. Software Engineering and Methodology*, 11(2), 191-214, 2002.

[2] Burnett, M., Cook, C., Rothermel, G., End-User Software Engineering, *Communications ACM*, 47(9), 53-58, 2004.

[3] El-Nasr, M.S., Smith, B.K., Learning Through Game Modding, *ACM Computers in Entertainment*, 4(1). Article 3B.

[4] Fielding, R.T., Taylor, R.N., Principled Design of the Modern Web Architecture, *ACM Trans. Internet Technology*, 2(2), 115–150, 2002.

[5] Huang, A., *Hacking the Xbox: An Introduction to Reverse Engineering,* No Starch Press, San Francisco, CA.2003.

[6] Henttonen, K., Matinlassi, M., Niemela, E., Kanstren, T., Integrability and Extensibility Evaluation in Software Architectural Models—A case study, *The Open Software Engineering Journal,* 1(1), 1-20. 2007.

[7] Narayanaswamy, K., Scacchi, W. Maintaining Evolving Configurations of Large Software Systems, *IEEE Trans. Software Engineering*, SE-13(3), 324-334, 1987.

[8] Leveque, T., Estublier, J., Vega, G., Extensibility and Modularity for Model-Driven Engineering Environments, *16th IEEE Conf. On Engineering Computer-Based Systems* (ECBS 2009), 305-314, 2009.

[9] Marino, P. *3D Game-Based Filmmaking: The Art of Machinima*. Paraglyph Press, Scottsdale, AZ, 2004.

[10] Parnas, D.L. Designing Software for Ease of Extension and Contraction, *IEEE Trans. Software Engineering*, SE-5(2), 128-138, 1979.

[11] Scacchi, W., Modeling, Integrating, and Enacting Complex Organizational Processes, in K. Carley, L. Gasser, and M. Prietula (Eds.), *Simulating Organizations: Computational Models of Institutions and Groups*, 153-168, MIT Press, 1998.

[12] Scacchi, W., Understanding the Requirements for Developing Open Source Software, *IEE Proceedings—Software Engineering,* 149(1), 24-39, February 2002. Revised version in K, Lyytinen, P. Loucopoulos, J. Mylopoulos, and W. Robinson (Eds.), *Design Requirements Engineering: A Ten-Year Perspective*, LNBIP 14, Springer-Verlag, 467-494, 2009.

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

[14] Scacchi, W. Free/Open Source Software Development: Recent Research Results and Emerging Opportunities, *Proc. European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, Dubrovnik, Croatia, 459-468, September 2007.*

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