Forthcoming in Games and Culture

What Can We Learn from Studio Studies Ethnographies? A 'messy' account of game development materiality, learning, and expertise

Jennifer R. Whitson
jwhitson@uwaterloo.ca

Abstract:

This article illustrates a gap between popular narratives of game development in design texts and the reality of day-to-day development, drawing from an ethnographic account of intern developers to highlight the potential contributions of studio studies to both game scholars and aspiring developers. It describes three take-aways. The first is that developers have difficulty articulating their work to others, impacting how we learn, teach, and talk about development, including how we share knowledge across domains. The second is that negotiation with technology rather than mastery characterizes the daily work of new developers, and the third is that problems frequently arise in articulating and aligning the normally black-boxed work of individual developers. Resolution of these issues commonly depends on 'soft' social skills, yet, external pressures on developers mean they tidy up and professionalize accounts of their daily practice, thus both social conflict and 'soft' skills have a tendency to disappear.

Keywords: studio studies, game development, ethnography, game developers, learning

For those interested in games and culture, we commonly learn by reading about game development, taking classes, and of course, making games. But what new things might we learn by watching game developers themselves at work? This article highlights a disconnect between how we might think games are made, and the messy realities of what team-based game development can look like on the ground. I tie this disconnect to written accounts of what development looks like, but also developers' own descriptions of their work. This reflexive narrative is prompted by my own first experiences in game studios, and the realization that none of the myriad accounts about game development had prepared me for the reality of team-based development, its daily ebbs and flows, and the challenges new developers faced. Perhaps unsurprisingly – this disconnect between expectation and reality was reflected in the experiences

of ten intern developers I shadowed in the early 2010s. Thus, this article explores three inter-related questions: What is left out from 'insider' accounts of game development and why? How does this impact how we learn about developers? And what new things can we learn through studio studies?

To provide background on how game scholars, such as myself, commonly learn about game development, I first review the material turn in Game Studies and the role of studio studies. I then outline my methodology, drawing lines around what I could – and could not - learn from an ethnography of intern developers whose practices differed from experienced professionals creating a game for commercial distribution. In the body of this article, I use a number of small vignettes from the field to address why learning about development -and how to become a developer- is so difficult. To do so, I draw from organizational sociology and studies of Science, Technology & Society (STS). These accounts focus on the situated, context-specific nature of learning, recounting the predominately mundane technical and social negotiation work that permeates everyday development but is missing from public accounts of development.

The discussion section reflects further upon why 'soft and squishy' socio-material accounts are largely absent from scholarly and industry discourses, and how the experiences of the interns studied link to more general understandings of game development. Not surprisingly, the reality of developing a game as a team is much more complex and messy than expected or planned for, and further studio studies may help us better understand recurring game design problems. The disconnect between discourse and daily practice has key implications, as what game scholars and educators think we know about game development shapes how we speak about game developers. We (re)produce discourses assessing what skills are valuable and taught, and who is valued in

gameswork and why. Commonly, these discourses focus on individualized technical expertise. This necessarily impacts games education training, the replicability of successful collaborative work practices, and developer culture.

Background and Context

Who is this paper for?

This article argues that social work, negotiation and collaboration play a fundamental but too little discussed role in game development literatures, and that this work is intrinsically tied to - rather than separate from - material and technological practices. These insights may already be familiar to instructors in game development programs, those with experience working in collaborative development teams, and students who are directly taught collaborative work skills in project-based courses. This article is for "everyone else who makes video games", a term borrowed from Brandon Keogh (2017). This term includes the majority of the interns I worked with, as even those planning to enter the formal game industry often avoid specialized game development degrees, and instead follow the advice of industry insiders to develop specialized skills and/or pursue "traditional" degrees (see, for example, Sinclair, 2014). Thus, for "everyone else" insights about the deeply social, often messy, nature of development are learned piecemeal, *in media res*, and often painfully. By emphasizing the social, I hope to spur more discussion and research on how to avoid common pitfalls and team implosions.

This paper is also for game scholars who, like myself, came from 'outside' formal game development programs. We enter the field by writing about game and gamers, but as a condition of employment we are increasingly asked to become "theorist-practitioners" and teach others to

make games.[1] This paper intends to incite more conversation about the skills we emphasize and impart. Finally, this paper is written for those who may have no desire to make games but are simply interested in games culture. Ethnographic approaches and studio studies are valuable to Game Studies, not only because they may help developers improve their craft, illustrating the varied contexts of and approaches to game making, but also because they highlight how both game scholars and developers approach their work with idealized preconceptions about game development roles, processes, practices, and values. These assumptions shape what game academics and game makers both see, how they talk, how they work, and the scholarly and cultural discourse around games that results. Drawing upon fieldwork, I suggest that the rationalized, often technical, individualized, and 'authorial' accounts predominating much of game development discourse, from textbooks to post-mortems to designer interviews and presentations, can miss where the real action is: the (often messy) socio-material negotiations between team members and their technologies.

Learning what Game Development Looks Like

To unpack pre-conceptions about development, it's important to first outline how game scholars commonly learn about developers, the strengths of these accounts, and what may be left out. Before gaining access to game studios, my own three windows into game development were textbooks on game design, developer postmortems, and academic accounts of game development. Each differently shaped what I thought development was supposed to look like.

Game Design textbooks

For most interested in game development, textbooks on game design are a primary resource. For me, this included Schell's (2008) *A Book of Lenses*, Fullerton's (2008) *Game Design Workshop*, Salen and Zimmerman's (2004) *Rules of Play: Game Design Fundamentals*, Koster's (2005) *A Theory of Fun for Game Designers*, and Bjork and Holopainen's (2005) *Patterns in Game Design.* In order to appeal to a broad audience, at differing levels of expertise, these texts offer an ideal, generalized blueprint and are largely technology-agnostic. Their lessons are applicable regardless of team size, game genre, development environment and tools, game engine and target platform. Universal takeaways predominate, although small case studies and reflections on designing on specific game features commonly add flavor.

Fullerton and Schell, in particular, helpfully outline development roles/tasks and core skill sets, mapping out the structures of teams and studios, and their connections from production to the larger games network of publishers, distributers, platforms and marketers. This clear and unambiguous mapping of what seem to be discrete roles, responsibilities, tasks and hierarchies in mainstream development framed how I thought game development worked: ideally, projects are driven by a designer's (or design team's) authorial intent and directives. Production is characterized by seamless coordination, individuals and disciplines working together (i.e. artists, programmers, sound engineers, designers), each enabled by domain-specific knowledge, assisted by sophisticated and powerful computer technologies, making individual contributions to a factory-like pipeline of content that becomes the final game. While essential orienting texts, these birds-eye view, by necessity, cannot linger on ground-level messes, conflicts and development dead-ends. Wanting detail on the ground level 'dirt' the textbook idealizations don't show, I turned to developer postmortems and other public accounts.

Postmortems

Game postmortems are written from the perspective of key members of the development team and reflect on what went wrong and what went right during the development process of a single game. Here, they offer insider detail on the messiness and contingency lacking in textbooks, focusing on perceived moments of conflict.[2] Despite the situated, project-specific detail, postmortems follow markedly similar scripts, citing repetitive issues with feature creep and scoping (attempting to create a game too large for the team size), scheduling (delivering the game on time), management issues, budget overruns, and the nearly universal crunch (extended, largely unremunerated working hours to meet deadlines) (Petrillo, Pimenta, Trindade, & Dietrich, 2009; Shirinian, 2011). The simple fact that these topics are commonly covered in texts, yet they continually recur, is a clear indicator that something more complicated is going on underneath the oft-repeated advice to "get better at scoping and planning". I argue that solutions lie in social, rather than technological, domains.

In their own analysis of postmortems, Petrillo et al. conclude that the focus on scope and feature creep indicate "traditional and game software industries do not suffer from technological problems, but from managerial ones" and that postmortems are "basically made up stories in everyday language" following similar, culturally approved scripts (2009, pp. 13.19-13.20). In short, postmortems tend to detail the same messes, and may not offer any replicable solutions. This is because postmortems, while ostensibly offering other developers a peek into their studio practices and problem-solving, are also a public relations tool used to attract players to the game, employees to the studio, and future financial partners and publishers. Thus, what is said within them is highly

structured and constrained. As noted by O'Donnell, important aspects of the development process are intentionally left out of postmortems: "Because many companies think that they might be giving away secrets, or because the documents must be cleared by legal departments, they frequently contain few details" (2009, p. 3.8). More personalized accounts of *what actually went wrong*, from feuds between team-members to days lost re-writing someone's sloppy code, may pinpoint root causes of scoping issues, but this airing of 'dirty laundry' jeopardizes one's professional reputation and future contracts.[3] Postmortem depictions of what game development looks like thus centre on time delays learning new technologies, relationships with publishers, and attempts to maintain a balance between ideal work hours, budgets, and delivery dates. These issues are illustrated to great effect by journalist Jason Schreier's (2017) account of ten studio projects. Because postmortems didn't tell me how developers and their technologies interacted on a typical day, I turned to a third source, studio studies and ethnographies of developers.

Scholarly accounts of development

The materiality of game development – and depictions of just how developers interact with each other and their tools – has not been a core focus of Game Studies until recently (Kultima, 2015, 2018). To help explain similar trends in wider scholarship, Wanda Orlikowski draws from Karen Barad to argue: "Language matters. Discourse matters. Culture matters. But there is an important sense in which the only thing that does not seem to matter anymore is matter" (2007, p. 1436). Materiality, in this sense, is treated as separate and distinct phenomena, as something to be considered only occasionally as specific technological events arise (i.e. 'special cases' such as the adoption and diffusion of a new technology, or spectacular failures and successes). When materiality is made a focus of study, generally we see research falling between two deterministic

poles. Techno-centric approaches are largely instrumental, assuming unproblematically that technology is largely exogenous, homogeneous, predictable, and stable, performing as intended and designed across time and place (as we see with many textbooks). Alternatively, human-centred approaches minimize the role of technology itself and focus more on the human side of the relationship, analysing how humans make sense and interact with technology, including their interpretations, interests and interactions.

In Game Studies there is a larger material turn characterized by interdisciplinary scholarship and an "increased concern for the contexts, uses and material qualities of games technologies on the one hand, as well as attentiveness to the situated analysis of play and players on the other" (Apperley & Jayemane, 2012, p. 7). Apperley and Jayemane divide this Material Turn into three intersecting bodies of literatures: ethnographies of games and players (see, for example, Humphreys, Fitzgerald, Banks, & Suzor, 2005; Taylor, 2006b; Chen, 2012), digital labour, particularly the study of user-created content, known colloquially as "playbour" (see Kücklich, 2005; Postigo, 2003; Dyer-Witheford & de Peuter, 2009), and platform studies. Platform studies unpacks the black box of game platforms, examining how the material computational limits of software and hardware platforms such as the Atari, the Wii, and Flash, shape and influence design decisions, thus impacting the player experience (see Montfort & Bogost, 2009). It thus aligns with the larger umbrella of software studies and the study of how 'cultural' and 'computer' layers of new media interpenetrate and influence each other.

Despite this material turn, platform studies methodologies are largely grounded in interviews, archival histories, and the deconstructions of hardware itself rather than day-to-day accounts.

Studies depicting the everyday reality of game development are relatively rare (see Banks, 2013; O'Donnell, 2014; Ash, 2015; Malaby, 2009; Whitson, 2017). Due to the secrecy of the game industry and concerns about inadvertent leaking of trade secrets, access to these sites is largely prohibited - particularly for humanities and social science scholars lacking programming skills (Nieborg, 2011). However, some researchers have had considerable success accessing smaller-scale, independent sites of production, 'indie' development (Browne, 2015). In particular, Fisher and Harvey's (2013) work on women's game-making initiatives and the conflict that arises within them illuminates some of the politically-charged realities of game production. This body of work contributes to a growing emphasis on studio studies, which focuses on the material and situated contexts of production, fostering a deeper understanding of how cultural artefacts are brought into being, and evidencing how creativity operates as a located practice (Farías & Wilkie, 2015). For example, Ash's ethnographic work (2015) describes how a 'fresh' and anxious QA tester clumsily playing a tutorial level for the first time on a monitor that just happened to be in view of senior designers spurred changes in a game section thought finished. In this sense, Ash argues that games, and by extension knowledge, do not arise from individual human bodies or brains, but are co-produced from these in-the-moment assemblages of teammates and technologies scattered across the office. Because these compositions are situated in space and time and overlap, the cultivation and export of specific desirable atmospheres to new domains is impossible: you cannot replicate the exact serendipitous atmospheres that contributed to a successful team dynamic and game, nor can you claim that authorship or inspiration is an individualized product. This may be one factor explaining why even established studios with senior developers struggle to replicate the success of a past game – the 'magic' combination of bodies/office atmosphere/technologies/context – could not be recaptured.

The shape and form of games is not only dependent on human developers, but also the interventions of material artifacts. Without that larger monitor, the above tutorial would have ended up very differently. Not surprisingly, development software plays a central role in ascribing status and expertise to humans. Rather than a mutely obedient tool, software exerts agency, allowing for -and even forcing- collaboration in the absence of human consensus. Software acts as a boundary object that unites different disciplines, aligning their practice in a shared direction, and thus altering the final form of games (Whitson, 2017). This work on developers and non-human actors parallels and complements existing scholarship on players, examining how game mods and bots exert in-game agency (De Paoli & Kerr, 2009; Taylor, 2006a). Games do not exist in and of themselves - they are not just rules embedded in hardware, but are also constituted by and operate in conjunction with players (Steinkuehler, 2006; Taylor, 2009). The resulting game is a mangling of human and material agencies (Pickering, 1995), just as game development itself is a constitutive entanglement of the social and material: a creative assemblage of game developers and the software they use, of white boards, headphones, sticky notes, and wikipages.

Methodology

Before moving to a case study illustrating what we may learn from watching developers at work, it's important to note that the specific study site and methods shape what we see. The vignettes for this article were collected during fieldwork in the summer of 2012 with ten intern developers at a large multinational game studio in Canada, which I refer to by the pseudonym, PlayHouse. The ten-week program was a collaboration between the studio and an academic funding network, who financed travel and lodging and assisted with student recruiting. The program was conceived with

two goals in mind: assisting student participants develop core competencies, aspirations, and careers in game design, and providing greater knowledge about interdisciplinary collaborative practices. According to PlayHouse's Creative Director, who led the program, rather than creating a commercial game the end goal was educational – experimenting with new internship models and educational partnerships.

The team was composed of three artists, three designers, three programmers, and one producer who I refer to using pseudonyms. Project roles were as follows: Jesse (2D art), Erin (3D art) Sam (rigging and animation), Max (narrative design), Nathan (game design), Alex (sound design), Martin (lead programmer), Kevin (system/tools programmer), Clark (AI programmer), and Tyrone (producer). While these roles are commonly found in game development, typically artists and programmers would outnumber designers. Participants each took on additional content development tasks beyond their primary roles, which were reflected in their final game credits. In terms of their previous training and experience with games, eight of the participants were working on or had just completed postgraduate degrees, while the remaining two had just completed their 3rd year of their respective undergraduate degree programs. Three had programing backgrounds, three had art and animation backgrounds, while the remaining four members had backgrounds spanning history, dance, cinema studies and linguistics. Most, but not all, had experience working on capstone game projects.

Unlike other internships where individuals are placed into apprenticeship roles and given more discrete tasks, the team was given autonomy: they were shown an existing game prototype, walked through it by the two designers, and were asked to use that game as a rough template, designing a

new game using the Unity game engine along with graphics software 3D Studio Max. PlayHouse provided office space, infrastructure (chairs, computers, software) and mentorship, including intensive training in the first week about how internal studio pipelines and production processes are structured, and group instruction sessions on communication and project management tools. Progress was evaluated by PlayHouse's Creative Director with weekly playtests and feedback sessions. Throughout the ten weeks, interns met one-on-one with senior programmers, artists, designers, and producers for more domain-specific guidance.

I and another ethnographer followed the project team for the entire period. Our research protocol included semi-structured entry and exit interviews with participants and PlayHouse employees, each of which lasted between thirty minutes and an hour. We also observed the development process, taking part in team meetings and scrums, sitting with individual team members at their desks, and noting their daily work flows. This was supplemented by access to various back-channel communications, including access to the team's Facebook, skype, and the project wiki page. Our observations took the form of field notes, audio and visual recordings, and photographs, and were coded and analysed using an iterative process throughout the ten weeks (Glaser & Strauss, 1967). I attuned to the ebbs and flows of everyday development, watching how developers learned and became professionalized in terms of how they spoke about their work and how they took on what they perceived as industry practices. This necessarily frames the interpretation that follows.

Studio studies commonly focus on commercial production practices within large, hierarchically organized professional teams (see O'Donnell, 2014), illustrating the pipelines and production practices for console and PC games, otherwise referred to as AAA production. However,

contemporary production practices have become increasingly heterogeneous in the networked era, exhibiting considerable variation in terms of team size and composition, business models, timelines and budgets, market, and infrastructure (Kerr, 2017). The so-called "democratization of development" facilitated by low cost tools and online distribution means that more and more games are made by “everyone else”: amateurs and hobbyists, students and indies, researchers and artists, operating outside of the commercial mainstream industry. These practices are not yet well-documented. Teams are often smaller, organized in a ‘flat’ rather than hierarchical manner, and multitask rather than occupy the discrete roles depicted in textbook organizational charts. In this study, the team was structured as a hybrid between traditional studio apprenticeships and the DIY autonomy of fledgling indie teams. While the team didn’t face the financial pressures endemic to commercial studios, other commonalities included the large scope, the tight timeline (which they met), and the sense that potential future employment rested upon their performance. In hindsight, if we were to evaluate the project in terms of educational and employment outcomes, after graduation, all ten participants immediately went on to work in games. Half a decade on, only one (Alex) no longer works in or around games. Tyrone runs a small independent studio, Erin teaches design at a college level, Clark, Nathan, Martin, and Kevin work at mid-size game studios, Max works at an interactive design firm, and Jesse and Sam are freelance artists. Thus we can assume that the project was at least, in part, successful.

From an STS perspective, studying interns rather than professional developers is a methodological artifact. It gives us access to developer culture in a way that is otherwise difficult - allowing us to see how aspiring developers struggling with mundane messy reality of work try and balance that reality with preconceived notions about what game development should look like. It’s important

to reiterate that this project set-up is not "normal" for professional studios. A large team of developers creating a game for commercial distribution with a mix of more senior and junior developers would be structured and behave quite differently, leading to different takeaways than the three I illustrate below. While the interns' experiences might differ from most large multinational studios, their struggles mirror those faced by growing number of small studios, particularly those composed of graduates unable to find employment in the mainstream industry. For example, the ESAC reports that there are 596 studios in Canada, growing 21% from 2015 to 2017. The vast majority of these (496) have 25 employees or less, emphasizing the rapid growth of small independent studios (Entertainment Software Association of Canada, 2017). Given the growth of studios of all sizes, much further work must be done to illustrate how practices differ from site to site, highlighting the range of different practices emerging beyond large-scale commercial development.

Snapshots from the Field:

Learning to become a game developer

In this section I begin addressing why learning about development (and how to become a developer) can be difficult, by using a number of brief vignettes from the field. I focus on three points. The first is that learning to make games is emergent, embodied and context-dependent, leading to challenges with teaching others and sharing knowledge. The second is that negotiation with technology, rather than mastery over it, characterizes intern developers' work. However, for better or worse, external pressures on developers mean they sanitize and professionalize accounts of their daily practice. The third is that problems frequently arise in articulating and aligning the

normally black-boxed work of individual developers. Resolution of these issues commonly depends on ‘soft’ social skills, otherwise referred to as heterogeneous engineering (Law, 1987).

The first vignette points to how learning is emergent, embodied and context-dependent. It is often difficult for developers to account for and explain their practice. This necessarily shapes how we learn, teach and even describe developer work and share knowledge across domains.

> This morning, Max is trying out rigging for the first time, adding what amounts to simplified bones and joints to the 3D model of his wheelchair monster, which later will be used to position and animate the monster. Sam, the animation and 3D art lead, is in the middle of a tutorial, talking to Max about how she's been using shortcuts for years, shortcuts that totally change how something works, simplifying and steamlining her work. But she can't tell Max exactly how to replicate them. She needs to show her, taking command of the computer while Max watches beside. Sam is too intuitive in her practice to explain the steps Max needs to go through. I pass by Max's desk a while later. She has a website open that walks her step-by-step through the rigging process. She’s pausing and following along with her own model. While Max still goes to Sam for trouble-shooting, this session marks their last ‘tutorial’.

Two things are illustrated by this example: 1) even in-the-moment Sam struggles to articulate what she is doing and why, and 2) for Max, hands-on learning is time consuming, but more effective than verbal and visual walkthroughs. This is in line with Orlikowski's (2002, 2007) work on how

knowledge is generated and shared in organizations. Orlikowski argues that knowledge and practice mutually constitute each other. “Knowing” is not some static property that can be separated from practice or simply embedded in some immutable mobile object (Latour, 1986), such as a game engine, or textbook, and transferred to a new community. To support her argument, she reviews multiple approaches to understanding knowledge, starting with Polanyi's (1966) distinction between tacit and explicit knowing. This points to a differentiation between ‘knowing how’ and ‘knowing what’: tacit know-how is the ability to put know-what into practice, and - unlike explicit knowledge- is often difficult or even impossible to verbalize and transfer to other users. For example, one may know the mechanics of how bicycles operate (explicit knowledge) without knowledge of how to actually ride the bike (tacit knowledge) and *vice versa*.

We could easily interpret Sam’s skill with ‘rigging’ the bones of the monster, but difficulty articulating exactly how to Max as a tacit, embodied, knowledge. However, Orlikowski argues that the focus on individually embedded tacit *vs* explicit knowledge limits other, more fruitful ways of thinking about knowing-in-practice. It is too individualistic to capture what is really going on between Sam and Max. Drawing from distributed cognition work and organizational sociology, Orlikowski suggests that ‘knowing’ is not a static embedded capability or stable disposition of actors, but an ongoing social accomplishment that is continually constituted and reconstituted by participants. As Schön puts it, "when we go about the spontaneous, intuitive performance of the actions of everyday life, we show ourselves to be knowledgeable in a special way. Often we cannot say what it is that we know. . . . Our knowing is ordinarily tacit, implicit in our pattern of action and in our feel for the stuff with which we are dealing," (as quoted by Orlikowski, 2002, p. 251). Thus, knowing is in our action. Expressing what developers “know” is prompted by interacting

with the technologies and team members around them, and isn't easily translatable or transportable to others.

Situated practice often involves reflection and experimentation, and how through such in-the-moment reconstruction of thought and action, knowing may be altered (2002, p. 253). Sam is very skilled in creating and animating 3D models with years of experience, but it is only though replicating her actual practices via the cues the material software provides that she can start to reflect and talk about what she knows. In attempting to teach Max, she cannot vocalize what she is doing. It is an intuitive and emergent process – a silent conversation between her and her software. Sam conducted her work without explicitly reflecting on the general principles or rules involved, yet performed rapidly and without deliberation when her hands are on the keyboard and drawing tablet. She struggled in expressing just *what* she was doing, and *why*, thus making the transfer of skills difficult as she could not distill intuitive actions into general rules, guidelines, or series of steps. Instead, she allowed the machine to prompt her, creating something start-to-finish, having Max follow along. To teach, artists on the team 'felt' their way through the software, haltingly verbalizing what they were doing each moment and thus developing guidelines along the way. In parallel to this "conversation with software" example, the narrative design of the game was also situated and collectively accomplished via bouncing ideas back and forth between three team members, who used prompts such as images, play-throughs and references to other games.

From this, it could be posited that knowing how to make games is inherent in action. Knowing is implicit, spontaneous and intuitive, and it is only achieved via making games themselves, a practice which we can see emulated in the larger pedagogical shift to 'experiential learning' and

capstone group project courses in education. In game development circles, this learning is encapsulated by the emphasis on game jams as skill development, and the oft-repeated advice to aspiring developers: 'Don't go to school. Just go make games' (see Stuart, 2014). It also means that lectures may be less effective teaching tools than software sandboxes and collaborative exercises. So, then, if knowledge is not something discrete and easily translated and transported, and if learning occurs via situated experience, what role does technology play in this process?

Collaborating with machines

Learning about game development can be difficult because so much of what goes on is a complex interaction with different sets of software tools. Game design textbooks are often technology agnostic, which might lead some to assume software functions as an obedient tool that can be 'black boxed'. However, software such as Unity 3D and 3D Studio Max actively shape not only the game, but how the team interacts with each other. This software becomes a site of social collaboration, but also of negotiation and resistance. A negotiation with, rather than mastery over, technology characterized the interns' work.

> Erin is standing at Sam's desk. Their increasingly loud conversation is starting to attract the attention of the rest of the ten-person team. On screen, a 3D model of a car door is slowly rotated as Sam investigates what appears to be a large divot in its center panel. Originally created by Erin, Sam downloaded the model from the server, making small changes. The divot appeared after Sam saved. Erin is furious: "*This is Sam's fault. There was nothing wrong with my model!"* Realizing the size of her audience, she sighs heavily, gets her bags, and leaves the office, stating that she needs a break. Sam, still moving sliders and manipulating variables to gauge

> their effect on the model, mutters in response: "*Erin is playing with options she doesn't understand"*. A few minutes later, after unsuccessfully attempting to de-divot the door, Sam camouflages the damage by applying the overlying texture, reflecting as she does so: "*It's a weird thing. Maybe even a bug in Max",* referring to the 3D Studio Max art software the team is using. *"It's not my fault. I swear to God".* The sound of typing begins again as the team returns attention to their screens.

Software tools are vital to collaboration processes, focusing and aligning the interests and productivity of team members who may not share a common goal or disciplinary knowledge. These tools channel agents' activities in pre-formatted directions, allowing for one's local understanding (i.e. an artist on a game development team) to be reframed in the context of some wider collective activity (i.e. collaborative game-making) despite the lack of a shared language, skillset or even consensus with other teammates on what the final project should look like. As such, software such as Unity and 3D Studio Max act as boundary objects (Star & Griesemer, 1989), that are plastic enough to be interpreted differently across communities, but contain enough immutable content to maintain integrity. Despite these tools being an obligatory passage point and uniting agent on the team, the software is often a site of breakdown, as with the inexplicable dent in the car door.

This messiness occurs for a number of reasons. Firstly, software such as Unity and 3D Studio max are not 'stabilized' after release. While marketing materials, game design texts, and online tutorials depict them as static and complete artifacts with a built-in array of fixed and predictable structures

and tools, the assumption of technical stability, completeness, and predictability collapses upon investigation. Bugs and glitches and breakdowns in functionality occur, particularly in alignment with updates and patches.

Secondly, while developers can and do use technologies such as 3D Studio Max as designed, they commonly circumvent inscribed ways of using the software—either ignoring certain properties and functionalities, working around them, or inventing new uses that may go beyond or even contradict the software creators' expectations and inscriptions. This is important because when we focus on embodied structures fixed in technology rather than looking for emergent technology structures enacted in practice, we conflate two very different aspects of technology: technology as an artifact that has material and textual properties – what 3D Studio Max is *supposed* to do versus what people *actually* do with the technological artifact in situated practice. Technology is not infinitely malleable, but how it is used in the field can differ greatly from its intended use.

Not surprisingly, the intern developers modified the technology in ways that are prescribed by the software designers (Akrich, 1992): for example, in Unity the programmers created additional layers of 'plug-in' tools that allowed designers to quickly modify the behaviours of enemy characters by toggling a few sliders. They also worked around software constraints, employing tricks and hacks in order to adapt available 3D technology to their needs. And so, the team's artists 'cheated' on most of the game's environment by quickly painting flat 2D billboards with trompe l'oeil depth indicators and dynamic lighting effects, creating a 3D illusion. As Monfort and Bogost (2009) show in their analysis of the Atari 2600, working around constraints in this way is a common practice. Social capital is accorded to developers who find new uses for hardware and

software, and coax more from machines than what was foreseen (or intended) by the system designers.

So far, the two vignettes can be read largely in terms learning to become a developer via situated practice, and learning how to collaborate with - as well as coax more out of - one's machines. As the following vignette demonstrates, the hardest-learned skills were deeply social: the ability understand the work of others, co-ordinate activities, and – in response to conflict – pull the socio-technical assemblage of people and their technologies back into some sense of alignment.

Heterogeneous Engineering in Game Development

During the course of the internship, problems commonly arose at articulation points where the normally black-boxed work of individuals was brought forth and integrated together. Effective resolution of these issues is dependent on 'soft' social skills. Key members of the team play an important role in finding alternative routes when the software resisted, employing numerous social and technical tactics and strategies to finish the game on schedule. Some actors have a more privileged relationship within the team, largely due to their perceived familiarity with the development software and their ability to coax it into alignment. Those, like Martin, are called on to shoulder an additional burden of work on top of their own assigned tasks as they channel and remove the 'gremlins' from other people's machines. Because of the interdependencies of artifacts created during the development process (e.g. code and assets), configuration management tools are used, allowing team members to work in isolation on one small part of the game (e.g. the AI for a non-player character, or a 3D model and texture for an in-game object) and then, once finished, re-integrate their product into the greater whole, ensuring that it doesn't 'break the build'

or create bugs. This is achieved using a number of additional software tools such as task managers, version control, bug trackers, etc., as well as software development protocols that prescribe the steps needed to properly reintegrate their work and ensure quality. It is at this reintegration point that things commonly go wrong.

> There are only a few weeks left in the program. Everyone looks tired. And bugs are everywhere. Martin is split between Jesse and Erin. Jesse can't figure out why there's a white outline on the wire mesh texture on his landscape, while Erin seated fifteen feet away, messages Martin on Skype, asking why she can't open the game level to test. After a few minutes and multiple Skype entreaties Erin heads to Martin's desk for a face-to-face appeal. Martin follows her back to her station. He is visibly frustrated, giving Erin step-by-step instructions on how to load a test level in Unity 3D. Erin is apprehensive about following them, and her tone is apprehensive.
> Martin: *Go into build settings.*
> Erin: *Why do I have to go there? I never go there?* (Her tone is questioning. This interface is new to her).
> Martin shows her how to use the menu, clicking on each element/level she wants to include in her test build. Once checked, Martin and Erin attempt to load the level. The entry screen runs, but nothing after that. The problem was not with Erin.
> Martin sighs. He asks Erin a question about her using her “repo”, which is shorthand for code repository. By this reference, he is indicating she should use version control software to access an earlier, working copy of the game until he has time to address

the problem. In response, Erin looks confused. She decides not to load the level, verbally noting that she has other stuff to work on.

Martin heads back to his desk, frustrated at the seemingly wasted tutorial. Sighing heavily, he looks up to the ceiling and shakes his head. He grabs his hair, which is long, dark, and tied back in a ponytail. This body language is like a whole-body eye roll aimed in Erin's direction. Visibly frustrated, he decides to drop what he's doing. *"I need to get back to the AI since it's holding everybody up and NOBODY bothered to tell me.*" Martin abandons his own tasks to return to the unfinished work that prevented Erin from loading a current version of the game. However, he soon interrupts this work to offer a work-around solution for Jesse, suggesting he use a shader on his art asset to minimize the appearance of, but not get rid of, the white outlines. More negotiation ensues as Jesse resists, stating that final version wouldn't look professional in his artist portfolio.

Later, when I ask Erin for clarification on why she gave up testing, she admits that she doesn't understand how the version control system works, but didn't want to take Martin's time to have him explain it. The team view is that Martin's expertise with Unity is precious. "You don't want to take more of his time than necessary, as he already has so much to do". Indeed, Martin stays in the office until nearly midnight, working hours after Jesse and Erin have left.

Martin's value on the team is tied not just to technological expertise, but to the ability to simultaneously visualize and coordinate the activities of different team members. Job specialization on a game development team means that members are able to 'black box' the work of others, not worrying about how individual elements are made and incorporated into the larger game, but still relying on the products of their labours. The better the work is done, the less visible it is to those who benefit from it. However, Martin's role as a 'fixer' means that he must bring such work forward, from art to programming, to design, and render it visible in order to solve technical issues. A large part of his time is spent on these efforts, rather than his own specific tasks.

Martin is performing articulation work (Strauss, 1988), moving from desk to desk and problem-solving in order to ensure that all the game assets can be taken from individual developers and successfully re-compiled back into a working version of the game. While balancing his own software engineer role, he is helping other developers troubleshoot, isolating the cause-and-effect sequences of what caused the unexpected software behaviour, and through trial-and-error experimentation demystifying recalcitrant machines and coaxing them back into alignment. However, the software often remains recalcitrant, leaving Martin to convince team members like Jesse to abandon resistant technology and instead find alternative routes.

John Law (1987) uses the term 'heterogeneous engineering' to refer to the social and material work required to arrange human and nonhumans into a stable artefact. Referencing Law, Lucy Suchman's ethnography of a bridge-building project evidences how even the most material artifacts (e.g. steel girders spanning a river) are constructed via persuasive performances. She details the social organization work done by various stakeholders who rely upon and reflexively

constitute all the different elements that need to be aligned in order for such a massive building project to take place: “The work of designing a bridge, on this view, is as much a matter of storytelling as of analysis, calculation, and work with concrete and steel” (2000, p. 311). In Suchman’s account, the material engineering work of designing and building a steel bridge (e.g. coming up with an efficient load-bearing design) is arguably a minor challenge compared to the social organization work of negotiating a solution that serves to protect local wildlife (from harvest mice to smelt fish), meets diametrically opposed budget, seismic, safety and traffic flow constraints, and enrolls local citizens who loudly reject the new bridge-building initiative as well as multiple municipal and federal works departments who have vested interests in alternative projects. In short, heterogeneous engineering is what resolved the largest challenges on the bridge project.

Like the work of designing a bridge, designing the digital spaces we inhabit can be as much a matter of storytelling and aligning diverse interests as it is of analysis, calculation, and work with code and pixels. Finishing this game was about keeping the wheels on the team: "Each one of us is an arrangement. That arrangement is more or less fragile. There are ordering processes which keep (or fail to keep) that arrangement on the road. And some of those processes, though precious few, are partially under our control some of the time" (Law, as cited by Suchman, 2000, p 311). Importantly, when boundary objects such as game software fall apart and fail to unite the social worlds of their members, then other means of creating cooperation arise, including silencing, fragmentation, coercion. Alternatively, social coordination work and heterogenous engineering are employed to re-align the team: allies are recruited and disciplined, while options are narrowed in order to get the team together and back on the road.

Martin, and the producer, Tyrone, brought the products of the different team members together in a way that created a more stable artifact. Other members struggled. They could readily master their own domain but ran into problems when they were asked to co-ordinate and collaborate their efforts, or work with other team members. Specifically, it was Tyrone that was attributed by others in their exit interviews for delivering a game on time and within scope, and holding the project and its members together. Heterogeneous engineers, with the abilities to bridge the different worlds of art, design, and programming worlds, aligning both technical and social, were thus valued by other team members. On paper, coming from a humanities computing background, Tyrone did not have any "essential" skills beyond sound design. By his own account, he took on the producer role as no one else wanted it. Along the way, he picked up any and all odd tasks that were behind schedule. Imminently approachable, the first person in the office and the last to leave, and quick to highlight his own shortcomings, Tyrone became the glue that held the team together, particularly when conflict ensued.

If we conceptualize knowing how to make games as an *enacted* capability, we see that game design competencies are not fixed or given properties that reside within developers. Rather they are constituted daily by ongoing and situated practices. Tyrone was the team 'MVP' not because of inherent programming or art or design knowledge, but because he was the right 'fit' for these specific team members. Keeping the end goal and weekly milestones in mind, he was able to interpret individual concerns, spot bottlenecks, pitch in and shepherd lagging members, and cajole or persuade those who had gone 'off track' back into alignment. Whereas developers such as Erin hesitated to "waste" Martin's time, they didn't hesitate to seek technical and moral support from

Tyrone. In this case, because he wasn't an 'expert' or specialist, other members didn't feel judged for not being experts themselves. With this team, being tech savvy was not as important as being able to communicate and problem-solve across a number of domains. While this may not be the case in a larger, more experienced development teams, the skills Tyrone demonstrated become increasingly important within smaller more agile teams, where roles, specializations and responsibilities are more blurred.

Discussion

While studying interns may limit what I can say about experienced developers, the focus on those entering studios for the first time illustrates the tension between what they thought development should look like (learned from textbooks, education, other developers, etc.) and the reality. The moments of social and technological drama captured in the vignettes detailed above were re-visited in follow-up interviews. I found a clear distinction between observing behaviours (ethnography) and spoken discourse (interviews): what interns said didn't always match what they did.

Problems were explained in terms of individualized properties (e.g. human error, lack of skill) or breakdowns in the system (e.g. bugs, equipment malfunctions) rather than failures in co-ordination or social conflict, presenting a rationalized account of what when wrong and why. For example, without fail, when asked to recount the problems and challenges of development, the interns – regardless of whether they were artists, designers or programmers, provided a list of steps they took to isolate, replicate, and find the source of technical problems. Challenges in development were depicted as technical, rather than social, and were solved via technical means.

Normative accounts of what the important problems are for development, and how to resolve them, represent idealizations or typifications, which delete contingencies and differences. These accounts serve important functions, for example, helping intern developers present a professionalized image of competency and thus find future employment. What is not said is as important as what is said. In this case, the silences in interviews about the social conflicts and struggles with technology that took up much of developers' time is telling. So, for example, in interviews, neither Sam nor Erin mentioned the vignette with the car door, their public falling out, nor any of the other struggles to get teammates and software to align. Sam started her interview expressing happiness there "weren't any personality clashes or differences of opinions, as everyone separated into their own teams". This oversimplification, or 'sanitization,' ensures that development issues are seen as structural rather than personal, thus preserving professional reputations and working relationships in a relatively insular industry

Game development work has a tendency to disappear, particularly social conflict. In response to Jerolmack and Khan's (2014) methodological provocation that "talk is cheap", it's important to emphasize that the disconnect makes interview accounts *more* valuable rather than *less*. Rather than being false or wrong, they illustrate how written and spoken accounts of development practice are inadvertently tidied up, hiding the 'mess'. Using these accounts, interns retroactively made sense of their actions, following scripts for what they think game development should look like, and in doing so inadvertently replicate some of the blind spots detailed game development literature.

Post-facto developer reflections on how development was organized stand not as explanations of the actual process, but as a product of, and resources for, team members' own ongoing (re)production and transformation of what game development is and is supposed to be. Expertise in game development relies also on social work, but in recounted narratives, including postmortems, it is technical expertise that is publicly foregrounded and used to ascribe status in development teams, because that is what expected from wider game development and game education communities. Extrapolating from Suchman (1995, 2000), learning how to be a competent game developer involves:

> learning how to translate one's experience, through acknowledged forms of speaking, writing and other productions, as observably intelligible and rational organizational action. Demonstrations of competence are inseparable, in this sense, from artful compliance with various professional and technological disciplines, reflexively constituted through those same demonstrations. (2000, p. 313)

For the interns, from what they read about online and in textbooks, and may learn in their courses, is that technological development is supposed to be scientific, rational and so this is what they depict in interviews and postmortems. Software is just a tool to be learned. Expertise is rooted in technological mastery, be it in design, graphics or programming. Messiness, including social conflict and skill-building, doesn't fit with larger cultural discourses of what game development is supposed to look like, and so it is largely ignored, thus replicating and perpetuating blind spots of our game development literature.

Importantly, while interns expressly framed problems as technical, when asked what they learned, responses were tied to social coordination, better understandings of how teams fit together, the

value of heterogeneous engineering work, and negotiation techniques. Tyrone was blunt: “People do whatever the hell they want. Game design is organized chaos”. Nathan paralleled this view, discussing the difficulty managing scope in terms of convincing team members to abandon tasks they were emotionally invested in, such as the inventory system. He notes he learned to be humble: “As someone who complains…complained all the time, I need to be more positive. I am more positive, now”. He continues “It’s like painting. All can do good. But to be great, you need to insist people do all these little tweaks. It’s all in the little details. It’s really hard to balance without looking like you’re complaining all the time…You choose your battles and you keep learning”.

Thus, while the soft and squishy-ness of technological work is an important part of any development practice, it often eludes examination. Sites of conflict, rather than barriers, are key learning moments, yet they are quickly tidied away in written and spoken accounts of development. It is, I know, unreasonable to expect game design texts, post-mortems, and academic writing on development to give an exhaustive account of what game development looks like. But from the vignettes and interviews above, I hope to have illustrated how difficult it is to speak about, develop and share knowledge about development practices without referencing the situated and social context in which games are made, and that by watching developers at work more often, we might shed light on why some team projects succeed while others implode, and start learning about what skills help teams function well together.

Conclusion

The interns’ challenges were less about design and mastering static technologies (what team members primarily would identify their work as) and more about the collaborative and distributed

nature of development work. This work is deeply enmeshed within an extended network of organizational activities such as sense-making, persuasion, consensus-building, and accountability. In short: figuring out how to put people and technology together.

Learning how to make games is ultimately situated, embodied and context dependent. This means, even in the moment, developers can have difficulty describing what they're doing or why. Because work is context-dependent (differing according to what game is made, where, by which team, and with what technologies) it is different each time. Thus, successful projects (by whatever metric chosen to measure this) cannot be replicated and translated *en masse*. This makes development difficult to write about and make generalizations from, thus partly explaining why textbooks cannot replace first-hand experience in game studios. This difficulty also shapes how we learn, teach, and even describe development work and thus share knowledge across domains. Not surprisingly, the reality of developing a game as a team is much more complex and messy than expected or planned for, thus contributing to recurring game design problems, such as scoping failure, feature-creep, budget and schedule overruns, overwork and interpersonal conflicts.

While intern accounts of development often emphasize technological mastery, in my own experience, negotiating with – rather than mastering – technology was a common occurrence, but one I did not expect: in order to develop a functioning game on time and feature-complete, intern developers quickly learned to hack solutions, try alternate routes, or simply disguise or abandon glitching assets, such as in the case of the car door. Drawing from later fieldwork with ten small studios and two games incubators, developers with more experience are speedier at recognizing familiar issues, finding workarounds, and identifying who they can go to address them. However,

negotiations with technology are not restricted to interns. For example, more complex architectures such as in-house engines that have been added to by multiple developers over time, or code written by developers long-since gone are specific sites of negotiation.

Beyond the technology itself, 'soft and squishy' skills were acknowledged as important to problem-solving and team functioning, however these same skills, and resources for building these skills, aren't as well-developed in the literature about game development. For example, while tutorials and texts on level design, game feel, AI programming, and marketing abound, there seems to be a lack of resources for aspiring game developers in terms of collaboration, coordination, and conflict-resolution. As more graduates of development programs launch teams of their own, they miss out on 'apprenticing' in established studios where they can watch and model the practices of experienced developers. Instead, they must learn by trial and error – and without the safety net of an employee salary.

Thus, while we may learn about development from textbooks, actual development experiences are far from textbook, requiring skills that are not often spoken about. An increased focus on the situated context of game development might help address some of these issues and direct more attention to skill-building in areas such as problem-solving, social coordination, and interacting within complex human-machine assemblages.

Notes

1. On a reflexive note, I did not set out to make games but along the way co-designed a locative mobile museum game, pitched in during fieldwork, and taught a number of undergraduate capstone development courses. While ethnographic access allows for a privileged view of development, I am not a developer.
2. While not relevant to this paper, previously, I systematically collected and coded Game Developer Magazine postmortems published from 2004 to 2011, analysing common development problems and how team size, budget, and development challenges related to specific game genres, consoles/platforms, and economic models.
3. Caroline Esmurdoc's atypical postmortems of Double Fine's Psychonauts and Brutal Legend significantly depart from this formula and make fascinating reads (Esmurdoc, 2005, 2009).

Acknowledgements

I would like to thank the two anonymous reviewers, as well as the editorial staff of Games and Culture for their comments in revising this manuscript. Any errors or omissions are mine alone. I would also like to thank Bart Simon, Aphra Kerr, John Banks, Casey O'Donnell, and Jason Della Rocca for inspiring work and conversations on studio studies.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Social Sciences and Humanities Research Council of Canada (grant number 756-2012-0209) and the Graphics Animation and New Media Network of Centres of Excellence (GRAND NCE).

References

Akrich, M. (1992). The De-Scription of Technical Objects. In W. Bijker & J. Law (Eds.), *Shaping Technology/Building Society: Studies in sociotechnical change* (pp. 205–224). Cambridge MA: The MIT Press.

Apperley, T. H., & Jayemane, D. (2012). Game Studies' Material Turn. *Westminster Papers in Communication and Culture*, *9*(1), 5–25.

Ash, J. (2015). Theorizing Studio Space: Spheres and atmospheres in a video game design studio. In I. Farías & A. Wilkie (Eds.), *Studio Studies: Operations, Topologies & Displacements* (pp. 91–104). London; New York: Routledge.

Banks, J. (2013). *Co-creating Videogames*. London; New York: Bloomsbury.

Björk, S., & Holopainen, J. (2005). *Patterns in Game Design*. Boston, Mass: Charles River Media.

Browne, P. (2015). *Jumping the Gap: Indie Labour and the Imagined Indie Community* (Master of Arts in Communication Studies). Concordia University, Montréal, Canada.

Chen, M. (2012). *Leet Noobs: The Life and Death of an Expert Player Group in World of Warcraft*. New York: Peter Lang Publishing Inc.

De Paoli, S., & Kerr, A. (2009). The Cheating Assemblage in MMORPGs: Toward a sociotechnical description of cheating. In *Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Retrieved from http://www.digra.org/dl/db/09287.23190.pdf

Dyer-Witheford, N., & de Peuter, G. (2009). *Games of Empire: Global capitalism and video games*. Minneapolis MN: University of Minnesota Press.

Entertainment Software Association of Canada. (2017). *Essential Facts About the Canadian Videogame Industry*. Retrieved from http://theesa.ca/wp-content/uploads/2017/10/ESAC2017_Booklet_13_Digital.pdf

Esmurdoc, C. (2005). Classic Postmortem: Double Fine's Psychonauts. Retrieved September 1, 2017, from /view/news/251220/Classic_Postmortem_Double_Fines_Psychonauts.php

Esmurdoc, C. (2009). Postmortem: Double Fine's Brutal Legend. Retrieved September 1, 2017, from https://www.gamasutra.com/view/feature/132696/postmortem_double_fines_brutal_.php

Farías, I., & Wilkie, A. (Eds.). (2015). *Studio Studies: Operations, Topologies & Displacements*. London; New York: Routledge.

Fisher, S., & Harvey, A. (2013). Intervention for Inclusivity: Gender Politics and Indie Game Development. *Loading - The Journal of the Canadian Games Studies Association*, *7*(11), 25–40.

Fullerton, T. (2008). *Game Design Workshop: A playcentric approach to creating innovative games* (2nd ed.). Burlington MA: Morgan Kaufmann Publishers.

Glaser, B., & Strauss, A. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New Brunswick: Aldine Transaction.

Humphreys, S., Fitzgerald, B., Banks, J., & Suzor, N. (2005). Fan-based Production for Computer Games: User-led Innovation, the "Drift of Value" and Intellectual Property Rights. *Media International Australia: Incorporating Culture and Policy*, (114), 16–29.

Jerolmack, C., & Khan, S. (2014). Talk Is Cheap Ethnography and the Attitudinal Fallacy. *Sociological Methods & Research*, *43*(2), 178–209. https://doi.org/10.1177/0049124114523396

Keogh, B. (2017, July 11). Who Else Makes Videogames? Considering Informal Development Practices. Retrieved July 11, 2017, from https://brkeogh.com/2017/07/11/who-else-makes-videogames-considering-informal-development-practices/

Kerr, A. (2017). *Global Games: Production, Circulation and Policy in the Networked Era*. London, UK: Routledge.

Koster, R. (2005). *A Theory of Fun for Game Design*. Scottsdale, AZ: Paraglyph Press.

Kücklich, J. (2005). Precarious Playbour: Modders and the digital games industry. *Fibreculture*, (5). Retrieved from http://journal.fibreculture.org/issue5/kucklich.html

Kultima, A. (2015). Game Design Research. In *Proceedings of the 19th International Academic Mindtrek Conference* (pp. 18–25). New York, NY, USA: ACM. https://doi.org/10.1145/2818187.2818300

Kultima, A. (2018). *Game Design Praxiology* (Doctoral dissertation). University of Tampere, Finland.

Latour, B. (1986). Visualization and Cognition: Thinking with Eyes and Hands. In E. Long & H. Kuklick (Eds.), *Knowledge and Society: Studies in the Sociology of Culture Past and Present* (Vol. 6, pp. 1–40). Elsevier Science Limited.

Law, J. (1987). Technology and Heterogeneous Engineering: The case of Portuguese expansion. In W. Bijker, T. P. Hughes, & T. Pinch (Eds.), *The Social Construction of Technological Systems* (pp. 111–134). Cambridge, MA: MIT Press.

Malaby, T. M. (2009). *Making Virtual Worlds: Linden Lab and Second Life*. Ithaca NY: Cornell University Press.

Montfort, N., & Bogost, I. (2009). *Racing the Beam: The Atari video computer system*. Cambridge, MA: The MIT Press.

Nieborg, D. B. (2011). *Triple--A: The political economy of the blockbuster video game* (PhD Thesis). University of Amsterdam, Amsterdam, Netherlands.

O'Donnell, C. (2009). The everyday lives of video game developers: Experimentally understanding underlying systems/structures. *Transformative Works and Cultures*, *2*(2009). Retrieved from http://journal.transformativeworks.org/index.php/twc/article/view/73/76

O'Donnell, C. (2014). *Developer's Dilemma: The Secret World of Videogame Creators*. Cambridge, MA: The MIT Press.

Orlikowski, W. J. (2002). Knowing in Practice: Enacting a Collective Capability in Distributed Organizing. *Organization Science*, *13*(3), 249–273.

Orlikowski, W. J. (2007). Sociomaterial Practices: Exploring Technology at Work. *Organization Studies*, *28*(09), 1435–1448.

Petrillo, F., Pimenta, M., Trindade, F., & Dietrich, C. (2009). What Went Wrong? A survey of problems in game development. *ACM Computers in Entertainment*, *7*(1), 1–22.

Polanyi, M. (1966). *The Tacit Dimension*. London: Routledge and Kegan Paul.

Postigo, H. (2003). From Pong to Planet Quake: Post-industrial transitions from leisure to work. *Information, Communication & Society*, *6*(4), 593–607.

Salen, K., & Zimmerman, E. (2004). *Rules of Play: Game design fundamentals*. Cambridge MA: The MIT Press.

Schell, J. (2008). *The Art of Game Design: A book of lenses*. Burlington MA: Morgan Kaufmann.

Schreier, J. (2017). *Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made*. New York: Harper Paperbacks.

Shirinian, A. (2011). Dissecting the Postmortem. *Game Developer Magazine*, (February), 7–12.

Sinclair, B. (2014, April). Should developers go to school? Retrieved August 31, 2017, from http://www.gamesindustry.biz/articles/2014-04-22-should-developers-go-to-school

Star, S. L., & Griesemer, J. R. (1989). Institutional Ecology, `Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, *19*(3), 387–420.

Steinkuehler, C. (2006). The Mangle of Play. *Games and Culture*, *1*(3), 199–213.

Strauss, A. (1988). The Articulation of Project Work: An Organizational Process. *The Sociological Quarterly*, *29*(2), 163–178.

Stuart, K. (2014, March 20). How to get into the games industry – an insiders' guide. *The Guardian*. Retrieved from https://www.theguardian.com/technology/2014/mar/20/how-to-get-into-the-games-industry-an-insiders-guide

Suchman, L. (1995). Making Work Visible. *Communications of the ACM*, *38*(9), 56–64.

Suchman, L. (2000). Organizing Alignment: A case of bridge-building. *Organization*, *7*(2), 311–327.

Taylor, T. L. (2006a). Does WoW Change Everything? How a PvP server, multinational player base, and surveillance mod scene caused me pause. *Games and Culture*, *1*(4), 318–337.

Taylor, T. L. (2006b). *Play Between Worlds: Exploring online game culture*. Cambridge, MA: The MIT Press.

Taylor, T. L. (2009). The Assemblage of Play. *Games and Culture*, *4*(4), 331–339.

Whitson, J. R. (2017). Voodoo software and boundary objects in game development: How developers collaborate and conflict with game engines and art tools. *New Media & Society*, *online first*. https://doi.org/10.1177/1461444817715020