The Effectiveness of Vertical Slicing In Game Development: A Research Project Proposal

Emily Bardwell
University of Colorado at Colorado Springs
Software Requirements 4310

4/11/2022

Introduction

The gaming industry is huge. It's an industry that generates billions of dollars each and every year. In 2018 alone the gaming market generated 134 billion dollars in revenue [9]. This is a huge amount of revenue and it's only predicted to grow as time goes on, which is why it is important to look at the game industry as a whole and how their processes work. There are many tools in the industry, but one particular tool stands out to me. The vertical slice is a development tool used in the gaming industry and is sometimes accompanied by an agile development schema. A vertical slice is like a small part of the game that is playable. This is similar to agile development used in other parts of the software industry, where you make small bits of working software based on client requirements once every couple of weeks. The main difference however is the frequency in which these bits of working software are created. Given the complexity of games a vertical slice doesn't always accompany an agile development schema. In some instances, such as in the development of the Saints Row series, a vertical slice is scheduled to take several months to build [1]. In other cases a vertical slice is paired directly with agile development and a new one is expected every 1-3 weeks [7]. In all cases it at least appears that the vertical slice is a powerful tool that is used frequently, but we must ask is it an effective tool. What, beyond creating a snippet of game play, does this tool tell us and can it help us?

Motivation

Game development is a complex process, which can take years to complete. During this time developers often face the issue of crunch time. Crunch time is defined as a period in development in which the developers have fallen behind on milestones for a particular project. During crunch time developers are generally expected to work longer hours under shorter deadlines in order to catch a project up to where it should be [2]. Though the term crunch time is generally used in the gaming industry I believe that it occurs throughout the software industry, and affects everyone, usually in a negative fashion. Crunch time costs time, a lot of money, and often more than not quality of the software degenerates. Developers are expected to do a lot of overtime. This can lead to exhaustion and burn out. Managers must pay their employees for this extra time worked and if the workers are entitled to overtime pay managers can end up paying an employee far more than is reasonable for the amount and quality of work done. With a combination of

making employees miserable, costing the development company money, and reducing the quality of software it is clear that crunch time should be avoided when possible and carefully managed when unavoidable.

Background and Related Work

The Vertical Slice

A vertical slice is a special tool in game development that somewhat mirrors the agile software development. In industrial agile environments we generally want to create a functioning bit of software, so that the customer may look and see how the software will work in the system. This is essentially what a vertical slice is. A vertical slice is a somewhat small working bit of software, which displays the game's core features. A lead developer of Volition, Greg Donovan, has spoken on how his company approached the vertical slice during the development of the Saints Row franchise. He described their attempt at vertical slicing in the first Saints Row game as a mix of success and failure. The vertical slice took several months to develop. They were working with a new platform, the xbox 360, a new open world mechanic, and several new technical tools for the development of the game. They previously had not done vertical slicing as their teams were smaller and it wasn't part of Volition's standards [1]. All of these factors come into play when looking at if vertical slicing is a useful tool, so we should keep them in mind for our research purposes.

A vertical slice can sometimes be accompanied by an agile development environment. An agile environment in the gaming industry is essentially the same as agile in industrial development. Developers work in one to three week sprints where they focus in on a feature or a part of a feature. Burn down charts are used and velocity is tracked in order to keep the team on top of the games needs. At the end of the sprint a vertical slice with the new feature is created [7]. This has gained traction in the gaming industry over the past few years, but it is unclear how many in the industry actually use it and use it in its pure form rather than a mixed methodology.

Technical Debt

Technical debt come in many different forms. In the paper 'An Exploration of Technical Debt' it is defined in 5 different major categories. The first is code debt. This type of technical debt is mainly about how well the code is written. Do the developers avoid duplication, is the code readable, and is it

organized are a few of the aspects of code technical debt. Essentially it is code that needs to be refactored. The second, design and architectural debt, is mainly about under focusing on maintainability of the system as a whole. The third type is environmental technical debt which is when we manually complete tasks which could be easily automated. The fourth is knowledge distribution and documentation debt. This type of debt is centered around how knowledgeable the developers are on the code base. New developers joining the team adds to this type of debt, while keeping old experienced developers keeps this debt low. The final type of debt is testing debt. This particular bit of debt is mainly caused by a lack of testing scripts leading developers to need to test the system manually, which takes quite a bit of time [3]. These different types of debt are extremely important to track. Each of them have different effects on the code base and should be tracked in order to ensure the technical debt of a project doesn't get out of control.

Crunch Time

Not much work has been done on the study of Crunch time. The paper "A Systematic Mapping Study of Crunch Time in Video Game Development" (2021) noted that nearly half of the papers they used for their study were student thesis and gray literature [10]. Further still most work done was in surveys, interviews, and mixed methodologies, rather than a more desirable observational study. Interview and surveys aren't bad, but with the human bias they aren't necessarily the best way to collect information on the main causes of crunch time. Many surveys and interviews find that employees are told crunch time is optional and the employees believe it is merely a matter of pride that pushes them to do crunch time, but I believe this is false. It isn't pride that pushes them to work the, sometimes extreme, overtime, rather it's the extreme social pressures and guilt thrusted upon the programmer which pushes them to delve into crunch time [4]. Everyone wants to produce good work and no one, absolutely no one, wants to let their team down. Those are their friends that they see and work with every day, so of course they want to do good by them. That's normal human behavior, but that doesn't actually tell us why crunch time exists.

Crunch time obviously has many factors leading up to it. One of the most obvious is bugs in the code, but it's not necessarily the leading cause of it. Bugs occur in all software projects but not all software projects suffer from crunch time, so bugs can't be the sole cause. Surveys of experts in the gaming industry have said that often at the beginning of development customers

are a bit unsure of what they want. This flippant behavior leads to crunching down the line as developers try to adjust to the different demands and requirements from the customer [5]. These last minute adjustments can cause extreme issues in terms of crunch time. There's also technical debt to consider. A poorly written code base and ignored tests have to be dealt with eventually and that eventually can lead to crunch time, but all these factors still exist in non gaming industries and are handled to some degree, so looking at what tools the gaming industry uses and how they manage their technical debts and customer requirements. For now though we shall delve deeper into why crunch time is an issue and what effects it can have on employee and employer.

There are many negative effects that crunch time can have on employees. The first is burn out. Burn out was first described by Freudenberger in 1974 as "to fail, wear out, or become exhausted by making excessive demands on energy, strength, or resources". Essentially it's when we use too much of our energy for work, and there are many symptoms one can develop when experiencing burn out. These symptoms are often split into 5 categories: physical, emotional, behavioral, interpersonal, and attitudinal. Under each of these categories there are a plethora of symptoms one might experience. Under the physical category burn out has been linked to poor general health which includes overeating, poor sleep, and gastrointestinal problems. The emotional category strongly links burn out and depression. Then under the behavioral and attitudinal categories we have the symptoms that hurt both the employee and the employer. These categories link burn out and high turnover, unproductive work behaviors, and negative attitudes towards ones job [6]. This is obviously not good for anyone. High turnover can cost businesses a lot of money and time as they struggle to look for suitable employees and then train them and catch them up with whatever project is currently on going. Having negative employees on the job site can lower overall moral which could just lead to more employees leaving, so burn affects far more than just the person who suffers from it.

Another major issue companies face with crunch time is the degeneration of work quality and overall productivity. When working overtime people often experience a symptoms of burn out called fatigue. In a study performed over 66 percent of programmers believed that mental fatigue had a large impact on the quality of their software. An observational study done in the same paper noted that with higher fatigue programmers are more likely to make mistakes in the quality of their code [11]. This is bad for both the program-

mer and the company. With lower quality code comes bugs, missing bits of features, and can cause difficulty later in development. This will make the users unhappy, and that can cost the company money as they try to fix the mistakes. The programmer likely doesn't want their code to be sloppy either. In fact it's a matter of pride for many that the code is nice and functional. Pride is what leads many developers to accept crunch time rather than push against it. They want their code to be nice and the users to be happy with it [10]. That's a good thing and we should all strive for high quality code, which is why we need to look at crunch time carefully when considering any development practices.
In order to properly pin down the causes of crunch time we need to take a closer look into the tools that are used and how effective they are. There hasn't been much research done on what tools are effective or not, so that's where we shall begin. In this paper we shall be focusing on one particular tool and how it might help or hinder the video game development process. This tool is called the Vertical Slice. - goes on to describe vertical slice.

Research Question

Is the vertical slice an effective tool in game development?
The vertical slice is a tool used in game development, but I'd like to know if it's effective and if so how effective is it and at what? In order to do this I'd like to observe the process as it happens in a real world situation. There are several factors that must be considered in order to obtain answers to the question and this question is kind of a big one, so I split it into 5 sub-questions.

1.Does Vertical Slicing allow us to accurately estimate the time it takes to develop a game?
Measuring this question would be relatively simplistic. We would want to measure the following for both the vertical slice and the finished game: how many features are present, how many man hours were put into debugging, how many man hours were put into documentation, and how many overall hours were worked. We want to look, not just at the numbers in a simple ratio comparison, but at the overall trends. It would be useful to then compare vertical slicing estimates to what truly occurred in the game's development. With the information on how long the game truly took to develop we can see how accurate these estimations are and how we might change them to be

more accurate.

2. Do investors care about the vertical slicing?
This question will center around publishing companies, investors, and crowdfunders. This means we will need to interview these people after being presented with the vertical slice and after the game's development is finished to see exactly how they react to it. A sampling of the questions I'd like to answer goes as follows:

1. Did the vertical slice help you get a decent feeling for how the game's development was going? Y/N.
2. Did you feel more confident in the direction of the game after seeing the vertical slice? Y/N.
3. Did you think of any unmentioned features that could be added to improve the game after viewing the vertical slice? Y/N
4. If so what ideas did the vertical slice give you? numeric estimation.
5. How many of those were implemented in the final game? none, some, half, many, all.
6. Do you want to see more vertical slices or is the amount the development company does appropriate? See less, see more, just the right amount.
7. Would you invest more money into this game given the vertical slice? Y/N.
8. Would you invest more money into the game if you hadn't seen the vertical slice? Y/N.
9. If yes would you invest the same amount, less, or more, and why? Short response.
10. Do you feel more confident about the development teams progress after viewing the vertical slice? Y/N
11. Would you be willing to change the release date after seeing a vertical slice if the developers think they will need it? Y/N

These questions are meant to gauge overall interest by investors due to the vertical slice and how that may affect the games overall funding. We would want to evaluate what the investors say with a heavy weighting on their interest in the game after the vertical slice and how much funding and leeway on time constraints they're willing to give the game after viewing the vertical slice.

3. How frequently do we catch gaps problems and inefficiencies when using a vertical slice?
This question would mainly involve interviewing developers, as they would be the ones keeping their eyes out for the given issues. It will be important to ask questions such as:
1. Were there any major inadequacies, ie gaps in the program or inefficiencies, you noticed during the development of the vertical slice? Y/N
2. If so, would these inadequacies be apparent if you had not done the vertical slice? All would, many would, half would, some would, none would.
3. If there were no vertical slice how long would it take for these issues to be caught? Short response.
4. Did the vertical slice help you understand what the game would be and how to improve it? Not at all, a little, or very.

The aim of these questions is to gauge how exactly the developers feel and interact with the vertical slice. This will give us an idea if it's useful for them in seeing the big picture and the little details at the same time, so they might catch major inefficiencies earlier in development rather than later.

4. Can vertical slicing help predict the number of bugs and man hours needed to deal with said bugs?
This would be one of the easier ones to evaluate, but it is extremely important. Bugs can cause all sorts of issues in a game and can make the user base very unhappy. There have been games that had to refunds to large parts of the user base due to the number of bugs in the game, such as Cyberpunk2077 [8]. For this we want to track both the number of bugs as well as the man hours taken to fix the bugs during the development of the vertical slice as well as the game as a whole. We should both compare the numbers as well as look for overall trends in development. I believe we are more likely experience trend lines in the amount of bugs and man hours it takes to fix them, because as a project gets more complex it becomes more prone to bugs and the complexity of bugs can rise as well. The result of course would be more bugs and man hours needed to fix bugs as time goes on in development. It would be useful to then compare these trend lines to other companies trend lines to see if there's a general trend that occurs. This should also be compared to companies that don't apply vertical slicing as a game development method, so we can see if vertical slicing has an effect on the number of bugs and the overall trend of bugs or not.

5. Can vertical slicing help predict crunch time or can it lead to it?
This question is by far the most complex. We must consider every factor that goes into development of a game as each one can affect crunch time. This question is also by far one of the most important. Crunch time leads to many issues in general and if we don't wrangle it in we could end up seriously damaging both people and companies. Some but not all of the factors we must consider, for both the game's overall development and the vertical slice, are: Overtime worked, man hours spent on technical debt, man hours spent on bugs, overall count of bugs, how big is the game, what type of game is it, how many lines of code is there, how experienced the team is in the vertical slicing method, are there previously existing systems the developers are reusing to speed development, are there any new tools the developers are working with that could slow development. All these factors and more would be boiled down to eigen values in order for easier evaluation on the whole.

In order to see just how useful the vertical slice tool can be it is imperative that we carefully go over each question and consider just how important that particular question is to the game development process as a whole. It is very important to keep investors happy, so that may rank higher than other questions in terms of usefulness. From that point findings can be presented to the industry.

References

[1] Greg Donovan. The vertical slice challenge, 2015.

[2] Henrik Edholm. Crunch time: The reasons and effects of unpaid overtime in the games industry.

[3] Richard Vidgen Edith Tom, Aybuke Aurum. An exploration of technical debt, 2013.

[4] Jan-Philipp Steghöfer Henrik Edholm, Mikaela Lidström. Crunch time: The reasons and effects of unpaid overtime in the games industry, 2017.

[5] Ruzica Jozin. Crunch time in software development: a theory, 2019.

[6] Sophia Kahill. Symptoms of professional burnout: A review of the empirical evidence, 1988.

[7] Clinton Keith. Agile game development, 2020.

[8] Cass Marshall. Cd projekt refunded 30k copies of cyberpunk 2077 out of 13m, 2021.

[9] newzoo. 2018 global games market, 2019.

[10] Jonne Niemelä. A systematic mapping study of crunch time in video game development, 2021.

[11] SAURAB SARKAR. Investigation of the effects of mental fatigue on programming tasks, 2015.