Development Methodologies in the Game Development Industry

A Thesis

Presented to the Faculty of

the Department of Computer Science and Information Technology

Kutztown University of Pennsylvania

Kutztown, Pennsylvania

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Aaron M. R. Harman

May 2023

Contents

List of Figures

1 Abstract

Video game development encounters unique challenges compared to other forms of software development. In addition to the management of programmers, game developers must also manage the creation of other assets to be used in development, such as art, sound, writing, and voice acting. The production of these assets has different management demands than production of code and must be accounted for by the development methodology of the project. Game development also is defined by vague requirements such as player enjoyment. As a result, the development methodologies of video games show changes and adaptations from those in software development to meet the change in demands.

This thesis will compare the methodologies used in software development to the methodologies used in the field of game development. The methodologies in use in the game development industry will be explored, in addition to how closely game developers follow the methodologies they claim to use. Finally, a methodology will be proposed for use by small game developers, based on the trends seen in software and game development and the unique needs of a small game development team.

2 Overview

2.1 Introduction

A video game is a piece of computer software produced with the intent to provide entertainment to the end user. After emerging in the 1970s, the video game industry has since grown into an enormous entertainment industry. While at first video game development could be accomplished by a small team or even a single developer, today's blockbuster games are created by teams of hundreds of people. This increase in size has necessitated the use of software development methodologies to coordinate that large of a team.

The development of a video game, while being similar to other types of software development in many ways, has different priorities than most types of commercial software. While all types of software development have to deal with changing project requirements, game development is especially prone to this problem. If software development can be compared to the creation of a tool, where the functionality of the end product is generally known from the beginning, game development would be more akin to a piece of art. The development of a game may change course based on the perceived success or failure of certain design decisions in meeting the expectations of the developer. For example, a mechanic that was intended to be a small part of the game may become more emphasized if it is found to be more engaging than other parts of the game. Since the notion of a game being ”fun” or ”engaging” are even more vague than the acceptance criteria for other software, producing an acceptable final game is much more reliant on continual adaptation.

This thesis will investigate the differences between software development methodologies used in the industry at large, and those used within the game development industry. The differences noted, as well as the demands of developing a video game, will then be used to propose a methodology for use in small video game development teams.

2.2 Definition of Key Terms

- **Video Game** - A piece of computer software produced with the intent to provide entertainment to the end user.

- **Indie Video Game** - A video game developed by a team without funding or support from a publisher, often produced by an individual developer or a small team. Some examples include Minecraft, Among Us, and Terraria.

- **AAA Video Game** - A video game developed by a team with funding and support from a publisher, usually produced by a large team, whose size may extend into the hundreds. Some examples include the Grand Theft Auto, Mario, and Pokemon franchises.

- **Asset Creation** - The process of creating art, textures, 3D models, writing, animations, music, sound effects, and other art, sound, or writing related resources during the course of game development.

- **Methodology** - A set of rules and processes to be followed while developing a software product.

- **Agile** - A software methodology that follows the priorities laid out in the Agile Manifesto, emphasizing individuals and interactions, working software, customer collaboration, and responding to change [1].

- **Iterative** - A software methodology that involves repeating some set of steps in a cycle, instead of following a linear process.

- **Game Mechanic** - One aspect, rule, or system, of a game. Examples include anything from player movement like running, jumping, or swimming, to other more complicated features like quests, weapons, scoring, or crafting systems.

2.3 Related Work

There are no similar published works that both analyze the developmental practices in the video game development industry as well as proposing a small games development methodology. However, there are some works that either analyze the development practices or propose a development methodology.

The studies that look into the methodologies currently used in the game development industry tended to find that iterative or agile methodologies were the most common, if any were used at all. Politowski et al analyzed the development postmortems of 20 games and found that iterative practices have become mainstream in the games industry [2]. McKenzie et al surveyed a group of game developers in New Zealand and found that while all their respondents claimed to use some kind of Agile methodology, many were not fully following the frameworks they claimed to use [3]. This is despite most self-reporting that they followed a methodology quite closely. Kasurinen et al also conducted a survey of game developers with similar results, finding that most developers used an agile methodology or nothing at all [4]. They also concluded that video game developers would benefit from a methodology supporting game development specifically. Stacey and Nandhakumar investigated the development process at a few mobile game companies, and found their process relied heavily on feedback from play-testing and the generation of good design ideas [5]. This included sometimes getting design ideas from people outside of the design team, and sometimes even outside of the development team. Kasurinen also found that while software and video game development share many features, there are some very clear differences between the two, especially concerning constantly shifting designs and more vague quality criteria [6].

There have also been a few publications that propose either partial or whole methodologies for developing video games. Taylor et al proposed a system for designing video games from a high-level perspective [7]. Marcelo and Davi proposed a system for video game development based off agile methodologies and incorporating Scrum [8]. Nandhakumar et al found that communicating a shared vision to the entire development team, across the various disciplines of which it is composed, was an important part of the design and development process [9].

3 Software Development Methodologies

A software development methodology sets out the rules and processes to be followed while developing a software product. Depending on the particular methodology, the details of what should be done and in what order may vary, but some general steps persist across all methodologies.

3.1 Software Development Steps

These steps are as follows [10]:

1. **Feasibility Study**

   In this step, the team investigates the technical and economic restrictions under which the project would be developed, and determine whether completing such a project would be feasible.

2. **Requirements Analysis**

   The team gathers information about the technical and personnel requirements for the project, which they then use to determine what resources must be allocated to this project's development.

3. **Design**

   The team produces documents outlining how the software product will operate once it is implemented. Depending on the methodology this may require designing the entire product or just a portion of it.

4. **Implementation**

   At this stage the development team uses the design that was produced to implement the software product using coding languages or other tools.

5. **Testing**

This step sees the team test the software produced for completeness and correctness. This can be performed using many different methods and may even be done synchronously with the coding step.

6. **Deployment**

   The product is deployed to be used, delivered to the customer, or made available for sale. This step varies depending on the type of software product and the business model used to get it to customers.

7. **Maintenance**

   This stage involves the company providing support for the software product after it is provided to a customer. This can include bug fixes, patches, feature updates, or other post-implementation changes.

3.2 Comparison of Methodologies

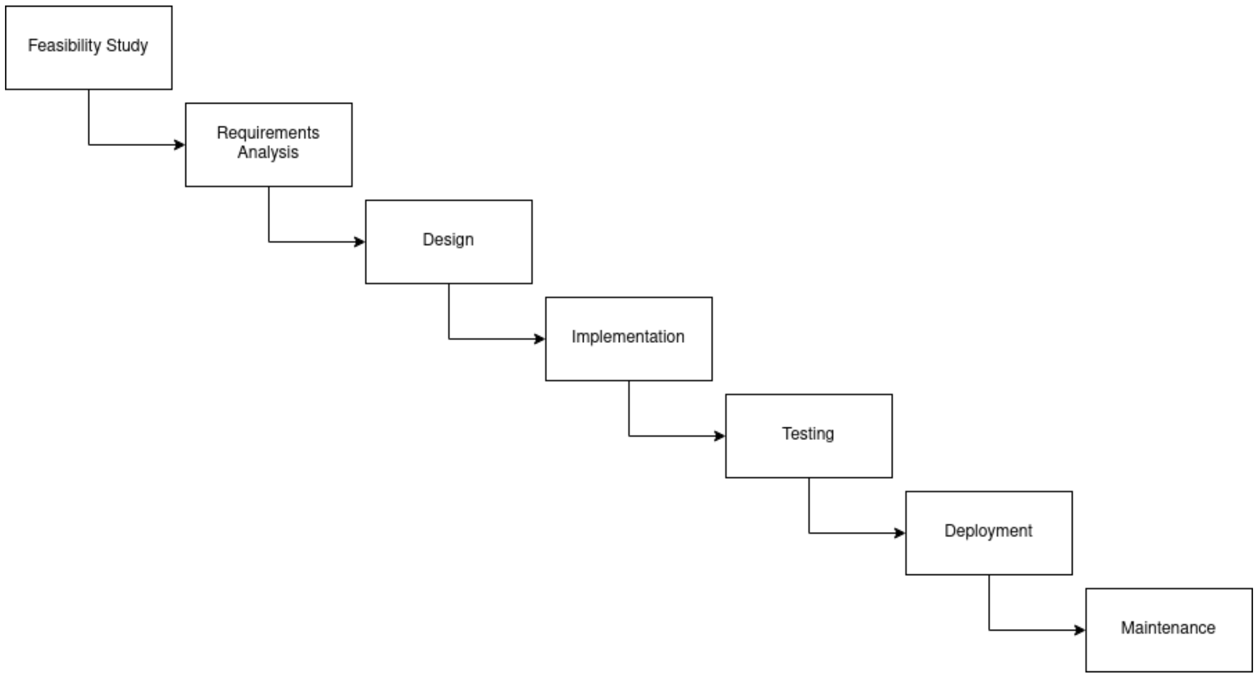

Figure 1: A diagram of a waterfall methodology process.

3.2.1 Waterfall

The waterfall method is the most straightforward of methodologies, where development proceeds one stage at a time, with each following stage not beginning until the previous stage is complete. Waterfall works best when the requirements for the project are well defined from the start and are unlikely to change during development. Many forms of waterfall do not have built-in ways to handle changing requirements, and as a result this methodology is not ideal for projects where the requirements are not clear.

While the ”pure” waterfall model does not have any mechanisms for handling change, there exist variants that attempt to fix this problem in various ways. For example, a variant of the waterfall model exists where each step ends by validating the results of that stage. If the results are not satisfactory, the development will not move to the next stage until any issues are remedied.

3.2.2 V Model

The V model is named as such because it pairs each stage of development with its own stage of testing, which are then gone through in reverse order. If there are flaws found during these stages of testing, development returns to the stage where the flaws were introduced and continues the process from there. The V model is more flexible than the waterfall method, but it still takes a while to catch and resolve issues in the software product.

3.2.3 Spiral Model

The spiral model creates a system for software development that attempts to minimize risk by incorporating risk evaluation into the process. The spiral model works in iterations, where each iteration starts with a risk assessment, which is used to determine what will be done during this iteration. Each iteration's tasks are determined by choosing steps that will resolve the major risk issues. At the end of each iteration, a review is performed. This review will change in structure depending on what was developed during this iteration, but its goal is to validate the results before moving on to the next iteration.[11]

The spiral model gives a framework for choosing what to do next but does not directly prescribe what must be done and in what order. It is even possible to break development down into multiple smaller spirals performed by different teams. The exact development path of two projects following the spiral model may look wildly different. The spiral model may become a waterfall model or an iterative model depending on the needs of the project.[11]

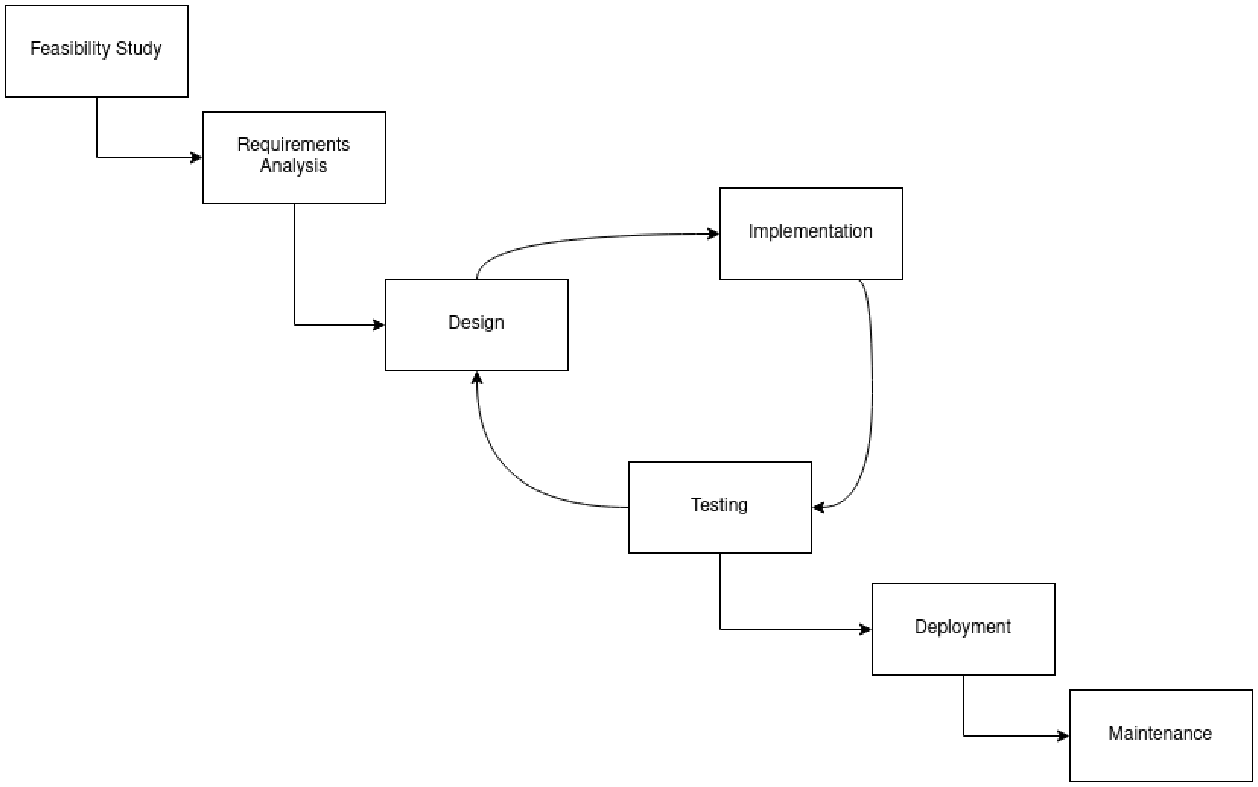

Figure 2: A diagram of an iterative methodology process.

3.2.4 Scrum

The scrum model is an example of an iterative methodology, where development is completed in small chunks called ”sprints.” Each sprint contains all the steps of a traditional software development method. Tasks for each sprint are pulled from a product backlog, which is a list of all tasks that need to be completed in order to complete development. These tasks are placed into a sprint backlog for the current sprint. The scrum methodology involves daily meetings, as well as meetings after each sprint, to keep the team on the same page and promote productivity. This methodology also includes a team role called the Scrum Master that makes sure the team is following the methodology correctly and removes any roadblocks in the way of the team doing so.[12]

3.2.5 Extreme Programming

Extreme Programming, often abbreviated as XP, is quite different from most other methodologies. XP decides what tasks will be prioritized by formatting the program requirements as story cards, which are then split into smaller tasks. These story cards are provided to the customer, who will decide which features should be prioritized. This system keeps the customer integrated with the development team, making it harder to lose track of requirements, and making problems apparent before they are difficult to fix. XP employs test-driven development, meaning that tests are written for all code before the code itself. The goal is to make sure that all code meets its requirements and requires that those requirements are clear before the writing of code begins. XP also utilizes pair programming, where developers are split into pairs, where one developer writes the code and the other watches and gives feedback. This practice is intended to increase code quality.[12]

3.2.6 Kanban

Kanban is a way of organizing the upcoming tasks in a visual, easy to read format. While Kanban is not a methodology of its own, it can be applied to other methodologies as a task organization strategy. This centers around the Kanban board, which breaks the tasks into three categories: to-do, in-progress, and complete. The board can be expanded to include more categories if necessary. This process produces a flow of development that can easily change along with changes in requirements or resources. Kanban also is useful for minimizing the amount of work in progress at any given time, thus increasing the efficiency of the team.[12]

4 Game Development Methodologies

A game development methodology sets out the rules and processes to be followed while developing a computer game. Since a computer game is a specific type of software product, the game development methodologies tend to be adaptations of software development methodologies.

4.1 Game Development Steps

The general stages for these game methodologies are quite similar to those of software development, and are as follows [13]:

1. **Planning**

   This stage encompasses everything from the idea conception to the pitching of the game idea to a publisher or team lead. The team produces a general concept of the game, as well as information like its genre and platform, during this stage.

2. **Pre-Production**

   This stage is analogous to the design stage of a software development project. The team produces documents outlining how the game will work, along with information like art and music aesthetic goals and game design priorities.

3. **Production**

   At this stage the development team uses the design that was produced to create art and sound assets, as well as implement the game using coding languages, game engines, or other tools.

4. **Post-Production**

   This stage is analogous to the testing stage of a software development project. In this stage, the team validates all the requirements of the game. This stage usually includes play-testing, where the game's design is tested by having people play through it as if they were an end user.

5. **Launch**

   This stage is analogous to the deployment stage of a software development project. This stage involves either sending a final product to be manufactured, or making the game available online, depending on the methods of distribution.

6. **Maintenance**

   This stage involves the company providing support for the game after it has been released. This can include bug fixes, patches, new features, or even paid expansions to the original game. The length of this phase can vary wildly depending on the type of game and the business model of the company.

4.2 Comparison of Game Methodologies

Game development methodologies are not well standardized, but the general categories of methodologies are as follows [2]:

- **Waterfall**

  This process is entirely sequential, with each phase starting after the previous phase is complete. This method requires explicit requirements that do not significantly change in order to be effective.

- **Iterative**

  This methodology develops the game in short cycles that go through all of the development phases. Each cycle adds new features to the game, until the game is deemed complete. This type of methodology can also be referred to as ”agile.”

- **Hybrid**

  This methodology is a combination of the waterfall and iterative methodologies. Some parts of the process are done sequentially, and others iteratively. Often, the waterfall model is used for the pre- and post-production, while the iterative approach is used for the production of the game.

- **Ad-Hoc**

  This is a catch-all category for any process made for an individual project. An ad-hoc process can take many forms, but usually is much more flexible than other methodologies due to the lack of clear definition of the process.

Iterative and Ad-Hoc methods seem to be the most often used in industry[2][3][4]. Game development is often defined by its ever-changing requirements, and as a result an iterative method would be a good fit for their development. An ad-hoc approach, despite its many potential problems due to the lack of structure, can also be effective due to its ability to be more flexible than any structured methodology. Unfortunately, it seems that many developers claim to be using an iterative or agile method, despite not using proper agile practices [2][3]. This casts doubt on the claims of studios who report using agile methods.

While game development shares some testable requirements with software development, such as platform compatibility and user interaction working as expected, it also has requirements characteristic of games that can make testing difficult. For most game projects, it is expected that they will be ”fun,” ”entertaining,” or ”compelling”[5][14]. These are criteria that are quite vague in their definition, and therefore cannot be tested for using any automated system or by writing tests before the software is developed. As a result, the testing of games is very reliant on putting a development build of the game in front of people who can play it and give feedback. This is known in the industry as play-testing[5].

5 Proposed Methodology

Based on the research into game developers and what methodologies they use, some trends have become evident. First, game development suffers from a tendency for requirements to shift significantly, which requires a methodology that can adapt to these changes. Second, usual methods of software testing will not be enough to determine whether the game meets its requirements, and therefore play-testing is needed. Lastly, teams have a tendency to attempt to follow a development methodology, but fail to implement it correctly, which will more frequently affect smaller teams who do not already have established development processes.

This section contains a proposed game development methodology for small teams. This methodology prioritizes allowing the design of the project to change, while keeping these changes structured enough to prevent the project from going off-course. This is a problem unique to game development and was a focus of this methodology. Considerations were also given for code reuse, to prevent wasting time or money on effort that will not make its way into the final product and synchronizing the creation of art and sound assets with the flow of development. This methodology is intended for small teams who do not have the support of a publisher, and as such deadlines imposed by an entity outside the team are not accounted for.

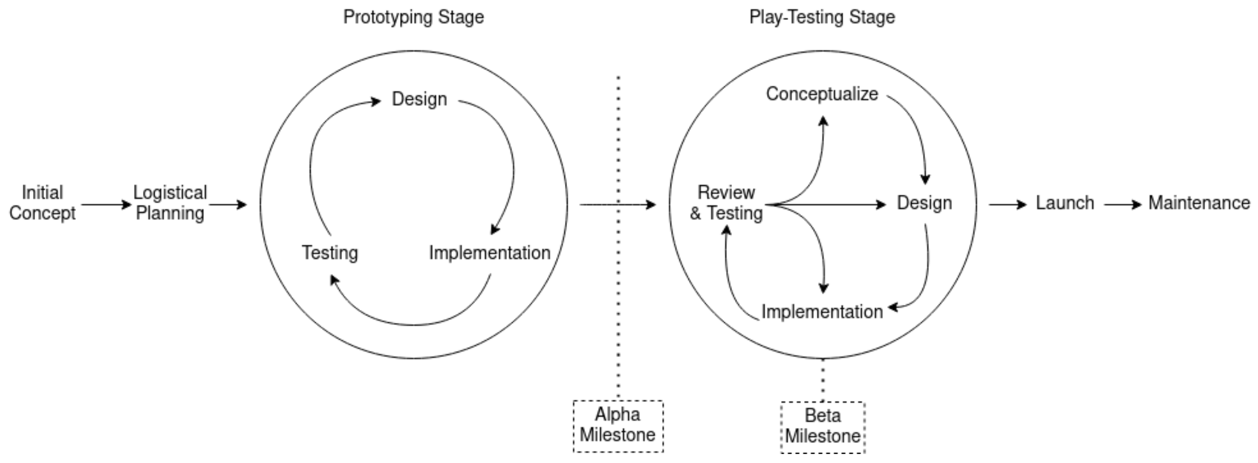

Figure 3: A diagram of the proposed methodology.

5.1 Planning Stage

This stage is quite similar to the usual planning stage of video game development. It is split into two smaller steps.

5.1.1 Initial Concept

The specifics of what occurs during this step will vary, as different teams will handle ideation differently. The main defining feature is that by the end of this phase, the team should have a general concept for the game they intend to produce. This idea is subject to change during the course of development, but will serve as the starting point for what is built in the prototyping stage. Important things to have decided include genre, platform, and scale, along with any other factors that would influence the tools the team would choose to use.

5.1.2 Logistical Planning

Once the initial concept is decided upon by the team, the next step is to determine how development will be structured and carried out. This step is where the tasks of the requirements analysis stage of software development take place. This includes making decisions about the eventual release model of the game, such as whether it will include expansions after launch or follow a rolling release schedule, as well as determining what resources are required for development and where they will come from.

During this stage the team should also determine what documentation should be made for each step of the development process. The steps in this methodology designate what decisions should be made by the end of each stage but depending on the structure of your game the best form to document these decisions may change. Documentation is not optional and must be utilized as both a tool for keeping track of design and development decisions as well as a written record of the development process for the benefit of an eventual post-mortem.

Once these broader decisions have been made, the team can decide how day-to-day operations will function. One part of this is figuring out what tools will be used for development. Deciding on these early is crucial, as changing tools later can be costly and time-consuming. There are many factors that will likely influence this decision. Some common ones will include budget, time constraints, team familiarity with the tools, and tool compatibility.

The other part of planning the structure will be determining the general team roles for each team member. In a small team, member roles tend to be flexible, with responsibilities fluctuating depending on the needs of the project, but it is still important to have a single team member who can ”own” each part of the project. For example, there should be one member of the team who is the owner of the visual direction. While other team members still have input on this part of development, this owner will be in charge of coordinating efforts and having a cohesive vision for their part of the project. This can be applied to many parts of the project, and these roles can change later if development requires it.

These two sections of planning should happen simultaneously. Determining which team members will be responsible for which part of the project will inform your decision on what tools to use, and your choice of tools will influence who on the team will be best suited to certain tasks. By the end of this step, you should be prepared to begin preliminary development work.

5.2 Prototyping Stage

This and the following stage take the place of the usual pre-production and production stages of game development. In this stage, development switches into an iterative process, where the goal of each iteration is to produce a prototype. These prototypes will begin as proofs-of-concept of various gameplay features or ideas, and by the end of this phase will become prototypes of multiple features together, up to prototypes of almost all of the intended gameplay elements.

This stage should not see changes to the overall idea of the game. For example, if your game begins this phase as a single-player first-person shooter with a focus on a zipline mechanic, your game should also leave this phase as that. The goal is simply to have gotten your initial idea to a point where it can be play-tested, and from there the idea can be tweaked as necessary. Changing the basic concept of the game during the prototyping stage would elongate the period before play-testing can occur but without any way to test the effectiveness of the changes since play-testing is not yet possible.

The following are steps that will be followed iteratively, beginning the process over each time it completes, until the Alpha Milestone is reached, which will be explained at the end of this section.

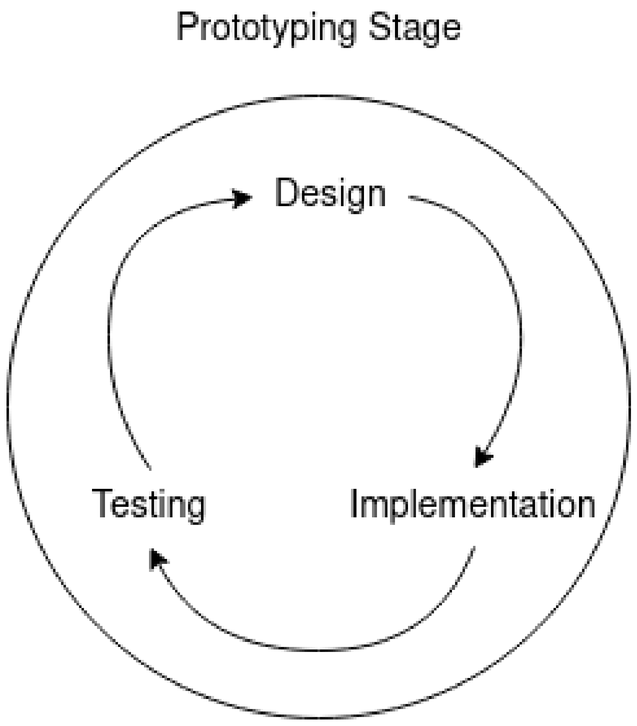

Figure 4: A diagram of the proposed methodology's prototyping stage

5.2.1 Planning

This iterative step will consist of determining what will be focused on during this iteration. Each iteration should focus on one mechanic or feature to be implemented, or possibly a collection of smaller related features. The size of each iteration should be such that the analysis of the results will be able to pick out bugs and possible design flaws with particular things implemented during it. When too many features are implemented at once, it becomes difficult to pinpoint which of these things is causing a problem that is noticed.

Each iteration need not be the same length, and the length of the iteration will also be determined during this planning. If the expected results are not achieved by the end of the iteration, the team can plan a follow-up iteration to finish these tasks.

This step also allows for the integration of another system like Kanban for planning out the tasks that must be completed for each iteration. Kanban is a good choice for combining with this methodology, as it gives a visual way to keep track of what tasks need to be completed, and also what tasks may need to be pushed to the next iteration.

5.2.2 Design

This step will involve the team designing the features that were decided to be worked on during this iteration. This step includes both the designing of a feature in game design terms, as well as desigining how the feature will be implemented in software development terms. This methodology focuses on the structure of how development should progress, not what game design principles should be used, so the contents of how this step is carried out are not prescribed.

5.2.3 Implementation

This step is where the designs are implemented using whatever tools were decided upon during logistical planning. The goal of the stage is to produce a working prototype of the design, so that it can be checked for design accuracy and bugs.

There are a few guidelines that should be followed during the development done during the Prototyping Stage. First, this code should be written in a way that is flexible enough to change the details of later. Keep in mind what values or behaviors may have to change later in the development process, and make sure they can be changed without unnecessary effort. For example, things like the speed of an entity or the numerical value of some object's statistic may need to change further in development to improve the game feel or to balance it with other alternatives within the system of the game. Development should be done with these changes in mind and accounted for.

Code must also be written in a modular fashion that will allow it to be reused after the Prototyping Stage. Unnecessary rewriting of code leads to wasted time, and as such all of the code written during this stage should be written with the intent of using it in the production build of the game.

Do not cut corners because this is ”only a prototype,” this code very well may be the code that is used to implement a feature in the release build of your game. How this manifests will depend on the tools your team is using, but it is an important priority in order to utilize development time most effectively.

5.2.4 Testing

Once your team has developed a prototype, the next step is to test that prototype. Depending on the scale of the feature that was being implemented, this stage may vary in its scope. The two main things that must be tested for are the code’s adherence to the design put forward, and a lack of bugs. Bugs can be tested for using any software development method that is applicable to the tools your team is using. The design adherence should be evaluated by the team as a whole, or by the team member in charge of the design of this component.

If the prototype is found to be lacking, then it should be given another iteration to iron out any problems. If the prototype is successful, then development can move on to whatever the team decides is a good next step.

5.2.5 Alpha Milestone

The ending point of the Prototyping Stage is reached when the prototypes built thus far have coalesced into a prototype of the full game that if placed in front of players to play-test could provide meaningful feedback. This meaningful feedback measure is the bar by which each iteration must be judged to decide if the team is ready to move into the next stage. Depending on the scale of the game, this may be achievable in a very short amount of time.

5.3 Play-Testing Stage

Once the alpha milestone is reached, development moves into the Play-Testing Stage. This stage is also iterative, but now instead of the goal of each stage being a prototype, the goal of each stage is a build of the game that can be put in front of players for feedback. This process will be gone into in more depth in the Testing section.

While this stage is still iterative, it is not iterative in the same way as the Prototyping stage. While the previous stage will always follow a circular cycle, where each iteration begins and ends in the same way, this stage follows a shape more similar to a boomerang. This is a method discovered in use in industry by Stacey and Nandhakumar [5]. This ”play-testing boomerang” ends each iteration at play-testing, but each following iteration may start at a different part of the process, depending on the direction the project is heading. Since game development is often closer to an artistic endeavor than a software project, giving the team the ability to change the nature and design of the project while development is ongoing is necessary, as if it is not accounted for in the structure of development, it will instead manifest outside of that structure and cause lost time and confusion.

Before the first iteration, the team will need to complete the planning step of the Testing step. Each subsequent iteration will end with Review and Testing, and begin in Conceptualizing, Design, or Implementation.

5.3.1 Conceptualizing

This step is unique to game development in comparison to software development. Software development projects very rarely see a change to the overall concept of the project, even if the design may change during development. Games, however, will sometimes entirely shift genre, or even general concept, during development. This is also a result of games’ similarity to art, and their vague overall requirements.

During this step, a feature or mechanic is either conceptualized for the first time as a new addition to the overall design of the game, or is reconceptualized due to problems or revelations during play-testing. This step encompasses the ideation process and the evaluation of the idea from a game design perspective to determine how to best fit it into the current design of the game. Similarly to the initial concept stage, this step is somewhat vague in structure, as different teams

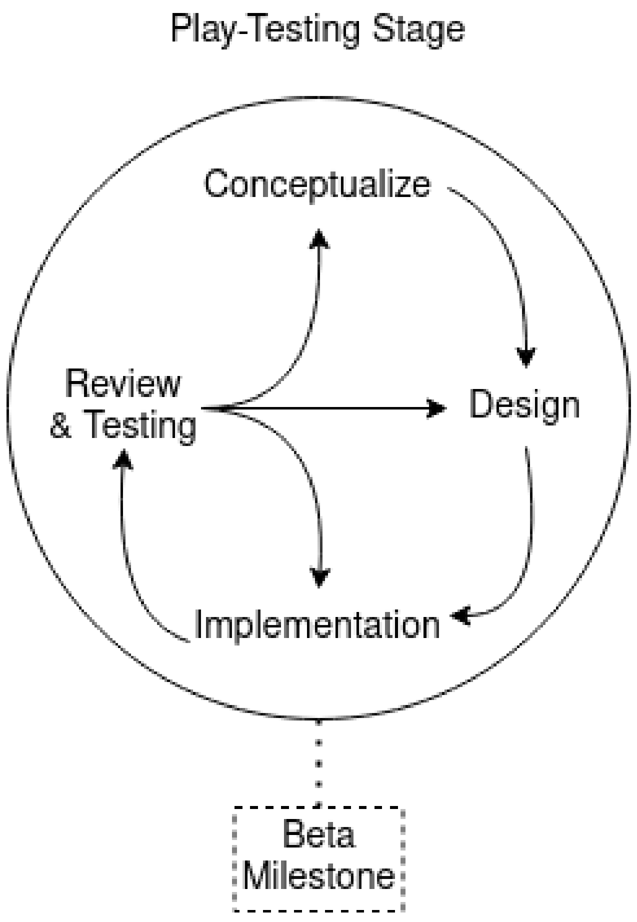

Figure 5: A diagram of the proposed methodology's play-testing stage

will generate ideas differently.

5.3.2 Design

The design step in the play-testing stage is very similar to the design step of the prototyping stage. This step does not include designing a feature from a game design perspective, unlike it's prototyping stage counterpart, as this is done in the conceptualizing step instead. In this step how the feature will be implemented is designed from a software development perspective.

Depending on the goal of the iteration, this step may instead be the design of a change to a feature or mechanic. In this case, this step will be spent determining the best means to make the desired change in the game. Care should be taken to make changes that can be reversed without unnecessary effort, and that leave the door open for more tweaks later in development.

5.3.3 Implementation

This step, like its counterpart in the prototyping stage, is where the designs are implemented using whatever tools were decided upon during logistical planning. The goal of the stage is to produce a playable version of the game, so the implementation should be done in a way that is complete enough to be tested by a user.

In the first iteration of the play-testing stage, this step will begin building off of the most full-featured prototype developed during the prototyping stage. Afterwards, each subsequent iteration will build off the previous one. Development should be done in such a way to allow future changes to features and mechanics without making large changes to the code. As with the rest of the process, future flexibility is always a priority.

Sometimes features that were not included in the most featured prototype at the end of the prototyping stage will still have prototypes associated with them. If development aims to implement that feature, the code from that prototype should be considered to see if it can be reused instead of writing the feature from scratch.

5.3.4 Review and Testing

This step of the play-testing stage is what drives each iteration. This step can be broken down into multiple tasks, some of which may not be performed every time.

First, the team must review the outcome of the development in this iteration. This is similar to the review process done in agile and iterative forms of software development. The team must determine whether their goal for the sprint was reached. If the goal was not reached, or a playable version of the game was not completed, then the team will skip the testing parts of this step.

If the build of the game produced by the end of the iteration meets the expectations of the team, and has added enough new content to the game that play-testing could result in helpful feedback, then testing should begin. How exactly play-testing should be administered is outside of the scope of this thesis. The users who are used for play-testing can be either people outside the team, or the development team itself. While using outside play-testers will usually give better feedback, as they are not as closely familiar with the game, using the development team itself as the play-testers can be preferred depending on the resources available to the team. After play-testing has finished, the team must review the findings of the testing so it can be used to inform future development.

From here, the team will plan the next iteration. Factors to consider while determining what to do next include how close the game is to completion, the feedback from rounds of play-testing, and outstanding design goals from previous stages. To streamline the asset creation process, tentative plans should be made for the contents of a sprint or two in advance. Since asset creation does not follow the same cycles as software development, having advance notice of what assets will be needed and when will reduce the need for placeholder assets while the real assets are being completed.

5.3.5 Beta Milestone

During the review and testing step of each iteration, the team must make an evaluation of the current progress of the project, and determine whether development should reach the Beta Milestone. After this milestone, large conceptual changes are not to be made, and it should be possible to plan out the remaining development work and project a release date. The decision of when this milestone is reached will be influenced both by the relative completion of the game and also the amount of resources remaining for the team.

While one of the priorities of this methodology is to allow for great flexibility during the development process, the Beta Milestone is intended to allow a more controlled finish to development. It should prevent large changes that will shift the delivery date of the game significantly past what is expected when the Beta Milestone is reached. This also should prevent the play-testing stage from continuing endlessly and never resulting in a finished product.

5.4 Launch and Maintenance

The launch and maintenance stages of the methodology will vary depending on the business model of the company developing the game. An example instance may have the game launched once, and then see periodic bug fixes or balance updates be shipped for the game after its initial launch. However, the launch stage may lead into more development, looping back onto the play-testing stage. How these stages will play out, and how the team will eventually wrap up development, should be decided during the logistical planning stage.

6 Conclusion

Video game development, while similar in many ways to software development, sees many differences in process due to different requirements. A game relies on vague requirements such as being ”fun” or ”engaging,” which do not allow for strict requirements testing. There is also the need to coordinate between members of the team implementing the game and those who are producing art and sound assets. As a result, the development methodologies of video games show changes and adaptations from those in software development to meet the change in demands.

It has been found that game developers prefer to use iterative, agile, or ad-hoc methods when developing games, but that they often claim to use a more structured approach than they do. The proposed methodology is intended to solve this problem by allowing the team to have the flexibility they need to make a good game while still staying within the bounds of a methodology. Future research could include application of the methodology to one or more game development projects and studying the results.

References

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development — agilemanifesto.org." `https://agilemanifesto.org/`, 2001. [Accessed 21-Apr-2023].

[2] C. Politowski, L. Fontoura, F. Petrillo, and Y.-G. Guéhéneuc, "Are the old days gone?," in *Proceedings of the 5th International Workshop on Games and Software Engineering*, ACM, May 2016.

[3] T. McKenzie, M. M. Trujillo, and S. Hoermann, "Software engineering practices and methods in the game development industry," in *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, ACM, Oct. 2019.

[4] J. Kasurinen, M. Palacin-Silva, and E. Vanhala, "What concerns game developers? a study on game development processes, sustainability and metrics," in *Proceedings of the 8th Workshop on Emerging Trends in Software Metrics*, WETSoM '17, p. 15–21, IEEE Press, 2017.

[5] P. Stacey and J. Nandhakumar, "Opening up to agile games development," *Communications of the ACM*, vol. 51, pp. 143–146, Dec. 2008.

[6] J. Kasurinen, "Games as software," in *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*, ACM, June 2016.

[7] M. J. Taylor, M. Baskett, G. D. Hughes, and S. J. Wade, "Using soft systems methodology for computer game design," *Systems Research and Behavioral Science*, vol. 24, pp. 359–368, July 2007.

[8] M. H. Marcelo and N. N. Davi, "Agile design: A combined model based on design thinking and agile methodologies for digital games projects," *Revista de Gestão e Projetos*, vol. 8,

pp. 109–126, May 2017. Copyright - Copyright Universidade Nove de Julho (UNINOVE) May-Aug 2017; Last updated - 2022-10-22.

[9] J. Nandhakumar, N. S. Panourgias, and H. Scarbrough, “From knowing it to ”getting it”: Envisioning practices in computer games development.,” *Information Systems Research*, vol. 24, no. 4, pp. 933 – 955, 2013.

[10] V. Chandra, “Comparison between various software development methodologies,” *International Journal of Computer Applications*, vol. 131, no. 9, pp. 7–10, 2015.

[11] B. W. Boehm, “A spiral model of software development and enhancement,” *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[12] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, “Empirical study of agile software development methodologies: A comparative analysis,” *SIGSOFT Softw. Eng. Notes*, vol. 40, p. 1–6, Feb. 2015.

[13] A. Aktaş and E. Orçun, “A survey of computer game development,” *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 13, pp. 239–251, Oct. 2014.

[14] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, “Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?,” in *Proceedings of the 36th International Conference on Software Engineering*, ACM, May 2014.