Accumulation and Management of Technical Debt in Video Game Development Artifacts

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Software engineering
June 2026
Riku Muuriaisniemi

UNIVERSITY OF TURKU
Department of Computing

Riku Muuriaisniemi: Accumulation and Management of Technical Debt in Video Game Development Artifacts

Master of Science (Tech) Thesis, 69 p., 2 app. p.
Software engineering
June 2026

In traditional software engineering, technical debt has been studied since Ward Cunningham introduced the concept in 1992. Unlike in traditional software development, less effort has been made to research game development empirically. The multidisciplinary nature of game development creates fundamental differences between traditional software development and game development. These differences raise an important question: what is the nature of technical debt in game development?

Technical debt in game development is explored through three questions: Why does technical debt accumulate in game development? What are the game development artifacts that contain the technical debt? And at last, what actions are taken to manage technical debt?

In this thesis, reader is first introduced to technical debt, and its management. Next, the taxonomy of game development artifacts is explored and adapted for the thesis, and then literature is examined for what already exist about technical debt in game development. Semi-structured interviews are conducted with five experts from the Finnish game development industry to answer the three questions presented.

Keywords: technical debt, game development, technical debt management, accumulation, artifact

Contents

List of Figures

List of Tables

1 Introduction

Video games differ from traditional software by intended use of it. Traditional software is developed to support operation of the company. Video games, on the other hand, are generally entertainment to the game's target audience. Requirements gathering for traditional software can be done by identifying what the customer actually needs. But how do you gather requirements for a video game? The main requirement of a video game to the user is for it to be fun. [1] And after all, definition of 'fun' can differ from person to person. In the article Murphy-Hill et al. say that developers may not have time to give effort on the design of game features, in the fear that the feature is not fun and it will be scrapped. [1] This may lead to inadequate design of architecture of the feature, architectural debt.

This is one of the many ways how video game development can differ from traditional software development. The differences in the industry's practices may affect attitudes, management methods used, and knowledge of Technical Debt (TD) in development of video games. What are the reasons for TD accumulation? What management methods are used to make sure that TD does not rule the development efficiency? And where does the technical reside in game development?

1.1 Motivation for Research

Software engineering has been researched for decades. Organized research and usage of the term 'software engineering' came into broader use from NATO software

engineering conference of 1968.[2] From the conference ever since new innovations in the field of software engineering has come to light, like Agile Manifesto in 2001 [3] and more recent innovations in the field of large language models [4]. These groundbreaking advancements by the industry professionals has been applied to traditional software development well.

But how has been these innovations applied to video game development? Murphy-Hill et al. state in the 2014 study that video game development and "traditional" software development have fundamental differences between them.[1] Authors also claim that "Videos games make up a significant part of modern software development, yet software engineering researchers in the past have made little effort to empirically study games." In fact, in 2023 global video games industry generated $183.9 billion U.S. dollars [5] while global music industry generated $28.6 billion U.S. dollars [6] and global movie industry generated $34.3 billion U.S. dollars at the box office.[7] Clearly video game industry generates a lot of revenue in comparison to other sections of the entertainment field.

1.2 Goals and Research Questions

The aim of this thesis is to identify reasons for technical debt accumulation, which artifacts contain technical debt, and how technical debt is managed in game development. First goal is to gain understanding what causes technical debt to accumulate in game development. For the second goal, the thesis aims to elaborate where technical debt could reside in and what technical debt types are common in game development. The point of the third goal is to gain insight to how technical debt is managed during development and what tools and methods are used. Three main Research Questions (RQ) are formed from these goals:

RQ1: What are main reasons behind technical debt accumulation in

game development?

RQ2: Which artifacts contain technical debt in game development projects?

RQ3: How is technical debt managed in game development?

1.3 Research Design

The thesis first introduces technical debt and relevant areas of game development to the reader through literature sources in Chapters 2 and 3. After the reader has been familiarized with technical debt and game development artifacts and previous research about characteristics contributing to technical debt in game development, the research of the thesis is conducted as qualitative research method semi-structured interviews in Chapter 4. Together with knowledge from Chapters 2, 3 and 4, the research questions are discussed in Chapter 5 and then answered in Chapter 6.

1.4 Use of AI in the thesis

Generative AI was used in the thesis in a supportive role to help author write grammatically correct text, better phrasing sentences, and helping translate accurate quotes from Finnish to English. All of the translations were reviewed by the author to be accurate to the best of author's knowledge. Generative AI was not used to generate original content for the thesis which was based on external sources and research done by the author of the thesis.

1.5 Structure of the Thesis

The rest of the thesis is structured in five chapters. Chapter 2 aims to explore technical debt, the types that are emphasized, and what constitutes technical debt management, including its activities and strategies. Chapter 3 elaborates relevant

areas of game development: game development artifacts, and what is already known about characteristics contributing to technical debt in game development. Chapter 4 details interview structure, interview participants and the results of the interviews. In Chapter 5, the results of the interviews are discussed together with previous research from Chapters 2 and 3. Finally, the thesis is concluded in Chapter 6, where research questions are answered, and limitations and possible future research are discussed.

2 Technical Debt

In 1992 Ward Cunningham first introduced concept what would later shape into technical debt:

> "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."[8]

Terms like "debt" and "interest" has been brought over from financial world to the world of software engineering by Cunningham.[8] Nowadays, technical debt does not only deal with code, but also debt accumulated during design or testing, for example.[9]

In this chapter technical debt is discussed in detail. The chapter will provide sufficient background information about technical debt for reader to understand what technical debt is, and why it is important. Section 2.1 introduces definition of technical debt used in this thesis and its components. Section 2.2 explores different technical debt types Finally, Section 2.3 explores three technical debt management frameworks.

2.1 Technical Debt Components

Much like the Cunningham's metaphor, technical debt terminology is based on financial field. Technical debt can be arranged into sub-components, which make the whole concept easier to understand: **principal, interest and probability of the interest.**[9], [10]

Shortcuts and sub-optimal solutions can sometimes be used in software development for short-term benefit. Amount of effort required to fix the sub-optimal solution, bringing it up to standard, is the **principal** of the technical debt. Principal repayment can be paid back by re-implementing the developed solution according to best-practices. Principal may not always be a shortcut in the code. Technical debt has multiple different types. For example, principal can also be deprecated documentation. More on the different debt types in Section 2.2. [9], [10]

If the module with debt principal is further developed or new features are developed on top of this already sub-optimal module, the principal will accumulate **interest**. Interest is the additional work caused by the principal. Interest can also manifest in form of decreased performance in development. To achieve the optimal state of the software, interest may also have to be paid. [9], [10]

Interest probability is where TD differs from financial world. In financial world, debts have interest which is negotiated in creation of the debt document. However, interest of technical debt does have a probability whether the interest must be paid or not. Interest probability means whether the interest will have an effect or not. Taking TD may not always affect project negatively, making the interest probability necessary part of the definition. [10]

2.2 Types of Technical Debt

Technical debt is not limited to only debt found in code. Different types of technical debt has been identified, and the types are defined by nature of technical debt. In 2013 article ”An exploration of technical debt”, Tom et al. identify five different types of technical debt: code, design and architecture, environmental, knowledge distribution and documentation, and testing debt. [9] Since then, set of TD types have been augmented. Initial study towards defining TD types by nature of TD is proposed in 2014 article ”Towards an Ontology of Terms on Technical Debt”. Rios et al. found that there are 13 different types of TD. [11] Article published year after found that there are 10 different types of TD, which all divide into several sub-types. [12] In the article ”Identification and management of technical debt: A systematic mapping study” Alves et al. expand the year 2014 article's dimensions of technical debt types from 13 to 15 different types. [13]

This thesis will use the 15 types defined by Alves et al. and focus on seven of them: code, design, architecture, test, test automation, documentation, and defect debts. The definitions for these types can be found in Table 2.1. These debt types have been selected based on relevance to this thesis and occurrence in development. [10] The rest of the types can be found in Table 2.2.

2.3 Technical Debt Management

Technical Debt Management (TDM) is set of activities that are used to identify and control the amount of TD accumulated during software development. There are also numerous strategies developed to manage TD and these strategies focus on the different activities of TDM. This section will first introduce the activities related to TDM and then focus on different strategies to manage TD.

Table 2.1: Technical debt types focused on in the thesis

T#	Type	Definition
T1	**Design debt**	Violating design principles of the chosen programming paradigm found in source code. [13]
T2	**Code debt**	Technical debt found in source code. Problems arise from complex, messy code differing from best practices and coding standards. [12], [13]
T3	**Architecture debt**	Problems in software architecture which affect internal quality aspects. [12], [13]
T4	**Test debt**	Issues arising from insufficient testing, e.g. non-maintained test suite. [12], [13]
T5	**Test automation debt**	Sub-type of test debt. Insufficient activities related to automating tests. [13]
T6	**Documentation debt**	Missing, out-of-date, or otherwise insufficient documentation in any aspect of the project. [12], [13]
T7	**Defect debt**	Known and tracked issues, defects, found in the software that, for some reason, will have to be fixed later. [13]

2.3.1 Technical Debt Management Activities

In a systematic mapping study, Li et al. compose set of eight TDM activities based on literature. The activities are as follows: identification, measurement, prioritization, prevention, monitoring, repayment, representation/documentation, and communication. [12]

Identification activities focus on detecting TD in the software project whether the TD accumulated is intentional or unintentional. **Measurement** activity does not focus on the amount of TD in the project but on the benefit of taking TD and what is the cost of TD in the project. Benefits scale with the principal amount and cost scales with interest amount. Before TD is repaid, it first must be **prioritized**. All of the taken debt is not equal in terms of impact. Debt that is estimated to have higher cost related to the benefit must be prioritized. After identifying and priori-

Table 2.2: Technical debt types exempt from the thesis

Type	Definition
Infrastructure debt	Problems in development-related processes, technologies, or tools that affect the development capability of the developers. [12]
Requirements debt	Completeness of requirements implemented in the software project. Deficient implementations are debt. [12], [13]
People debt	Issues rising from human resources in the project. It can source from lack of training or expertise.[13]
Build debt	Overly complex and non-optimized build process, that is slow and difficult. [12], [13]
Process debt	Development processes that give short-term benefits, but affects long-term negatively. [14]
Usability debt	Development decisions that affect usability of the software negatively. For example, lack of usability standard. [13]
Service debt	Poorly selected and utilized services for software project in service-oriented architecture. [13]
Versioning debt	Problems with source code versioning. [12], [13]

tizing TD items, they can be **repaid**. TD items with highest priorities are the first ones to be repaid. The benefit and the cost of TD can change over time. Further development of artifacts related to taken TD can change the benefit and the cost of TD item from when it was first identified and measured. To detect the change of the TD and thus priority of TD, it must be **monitored**. Accumulating TD is not always beneficial if the cost of TD outweighs the benefits of it. TD **prevention** is important TDM activity that will focus on preventing any unwanted TD. The last two activities, **representation/documentation** and **communication**, are about establishing understanding of the TD state to all stakeholders. Representation/documentation is about representing the TD in clear and concise manner to address stakeholder concerns. Communication makes identified TD visible to stakeholders

and thus enabling communication. [12]

Three of the activities have received significantly more attention than the other five; repayment, identification and measurement are studied in 59, 51 and 49 studies respectively while rest of the activities have less than 20 studies each. The least studied activity is representation/documentation with 4 studies. The three most researched activities are obvious choices: they are the core activities of TDM. [12]

2.3.2 Technical Debt Management Strategies

In addition to activities, multiple different strategies has been developed for TDM. Based on the mapping study by Alves et al., two of the most commonly presented strategies in literature will be presented. The strategies are cost-benefit analysis and portfolio approach. [13]

Cost-benefit analysis is a strategy to determine whether identified TD is worth paying off. A list of technical debt items (TDI) is formed by identifying the items. The TDIs are given a description which details where the debt has been accumulated and estimate of interest and principal amounts. Interest and principal is given first a coarse estimate (low, medium or high) and later they are more finely estimated when more information has been gathered. Then it is measured which items are worth paying off based on the interest and principal amounts. Items can be arranged in a matrix where y-axis is interest amount and x-axis is principal amount. High interest means higher impact if interest must be paid too.[15] Example matrix is shown in Figure 2.1. Seaman et al. in the article "Using technical debt data in decision making: Potential decision approaches" suggests debt items located in the left upper corner are worth prioritizing. Items located in there have higher interest and therefore impact, but the principal is low, requiring less effort to pay it off. Next items should be selected by moving to down and to the right in the matrix. [15] Following Seaman et al. advice, items in the matrix shown in Figure 2.1 should

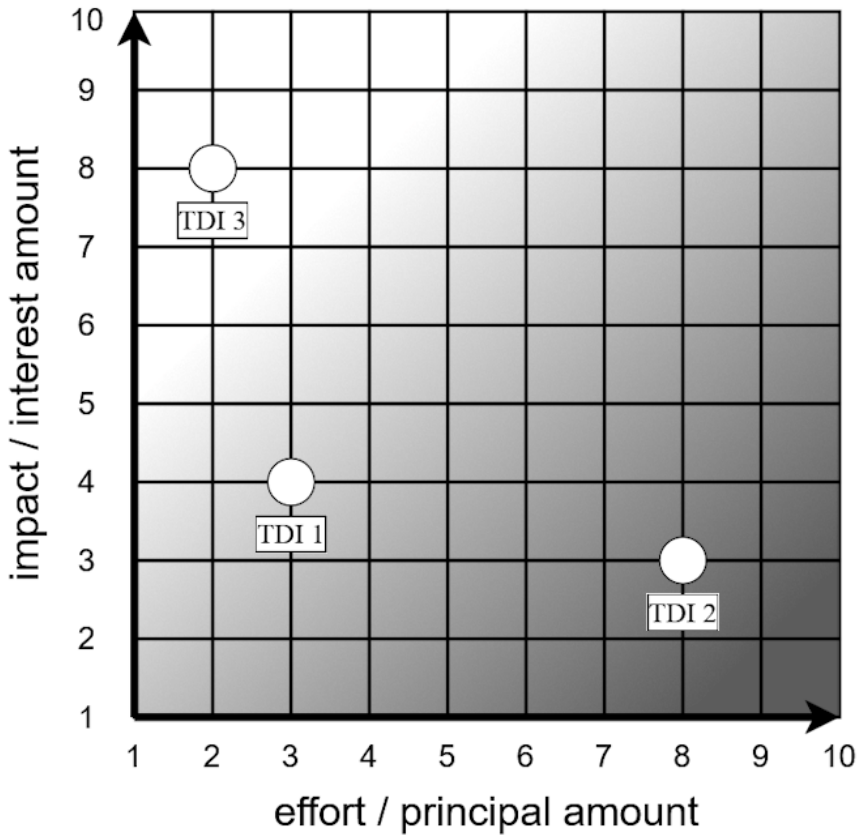

Figure 2.1: Example of cost-benefit matrix

prioritize paying off TDI 3 first, then TDI 1, and TDI 2 at last.

The portfolio approach by Guo et al. use similar technical debt list and items as the cost-benefit analysis. In the portfolio approach the TDI are a lot more detailed. The items have detailed description of the incurred debt. Description includes the location of the debt, identification time, who identified the debt item, why is the item considered debt, principal estimation, expected interest amount (EIA), interest standard deviation (ISD), and if there is any correlation with other TDI. ISD is used to trying to predict the uncertainty of interest amount. Correlation between other TDI might affect item's interest amount. First the TDI is identified and the item listing is populated with metadata related to discovery of the TDI. Then it is time to measure the impact of the TDI. The creator of TDI will roughly estimate principal, EIA, ISD and TDI correlations with their own experience. Principal, EIA and ISD

estimations can be fine-tuned with data from previous experiences. Correlation fine tuning can be more difficult: it may need analysis of system architecture or dependency analysis. To help with the decision process of whether the TDI will paid off, net benefit of each item is calculated by subtracting interest amount from the principal. Risk of not receiving the benefit (interest and principal needs to be paid off) is called variance of return. Finally, a decision of paying back debt is made during the planning of next release. Affected module's TDIs are listed and re-estimated according to the new features to be developed. Then a risk level is calculated for the items. New portfolio is calculated and items left out of the new portfolio need to be paid back before the next release. [16]

3 Video game development

Video games have evolved massively since the first video games like "Tennis for Two" (1958) and "Spacewar!" (1962) [17] to more recent releases like Battlefield 6 released in 2025. During the evolution of video games, new controllers, genres, and ways to play have emerged while the scientific research has fallen behind the industry.[18] Chueca et al. state that research on Game Software Engineering (GSE) is growing more than ever. In their study they show that 74 industrial studies out of 98 in their dataset has been conducted between 2009 and 2021. The remaining 24 studies have been published before 2009. Comparing quantities show that industry-scale research has grown over four-fold after 2009 compared to before 2009, which is a great increase. [18] Chueca et al. conclude in their study that traditional software engineering (SE) and GSE are diverging to separate domains and GSE is becoming its own, independent domain [18]. These differences between domains might lead to differences in technical debt features.

This chapter acts as basis to understand game development artifacts and potential characteristics contributing to accumulation of technical debt in game development. In the first section, taxonomy based game development artifacts will be introduced. In the next section, focus will shift to what characteristics are already known to contribute to technical debt in game development. These characteristics are then examined further to find out what are potential technical debt types rise from them and how these characteristics could be linked to game development

artifacts.

3.1 Game Development Artifacts

Game development process produces a variety of different artifacts. Lack of standardization and disorganized creation and maintenance of artifacts is common in the industry. Challenges in understanding game development artifacts may affect the understanding of technical debt in the artifacts and presenting it. This thesis will use the taxonomy of game development artifacts created by McDonald et al. in "Describing, organizing, and maintaining video game development artifacts", to clarify which artifacts are affected by technical debt and enhance the understanding. The taxonomy has three broad sections: Development, Organization-Related Artifacts, and Marketing. Upon investigating the taxonomy[1] and its artifact descriptions, only the development section has items that could be related to technical debt. The development section contains artifacts that are related to design and implementation of the game itself. The section is further split into seven different subsections: Design, Engineering, Quality Assurance and Control, Art, Audio, Writing, and Production. [19]

This section will present the artifacts from the taxonomy. The artifacts are presented in subsections of their own and summarized in tables. As this section is meant to act as introduction into the game development artifacts and the large number of them, the taxonomy is adapted to present the most relevant artifacts. This presentation will group related artifacts and omit those that are included into other artifacts or if they are not relevant to the thesis' topic.

Table 3.1: Design artifacts adapted from the taxonomy [20]

A#	Artifact	Description
A1.1	**Brainstorming Document**	Notes or records of ideas regarding early design and concept of the game.
A1.2	**Concept Document**	Overview of the game concept. May explain characters, gameplay, genre, or plot.
A1.3	**Game Bible**	Collection of information about game's concept, development, process, and structure. Information to help with design choices.
A1.4	**Game Design Document (GDD)**	Living document used to communicate game design elements to the development team. Ideally always up-to-date to represent latest state of a game.
A1.5	**Research Materials, Consumer Research Materials, Reference Materials**	Collection of materials intended to guide game design. Consumer Research Materials contains information regarding target audience of a game. Reference materials are used to inform implementation used in game by providing reference media.

3.1.1 Design

Design artifacts are related to overall conceptualization and design of the game elements. Artifacts in design subsection are presented in the Table 3.1. Brainstorming document is created when first brainstorming for early design of a game. Concept document is an overview of a game that tries to explain main features of characters, gameplay, genre, or plot. Game bible is document that tries to outline game features using concept work from all disciplines of game development. In some cases, term game design document is used when referring to game bible. Game design document is constantly evolving document that communicates design ideas to development teams. Game design document should be updated to reflect latest state of a game. Research materials are media of any type used to guide game design. Research materials include consumer research materials which should provide in-

[1]The taxonomy is available at `https://github.com/uwgamergroup/taxonomy-game-development-artifacts`

sight to the target audience of a game and reference materials that should guide implementation of game elements by providing reference point, for example pictures of a famous building. [20]

3.1.2 Engineering

Table 3.2: Engineering artifacts adapted from the taxonomy [20]

A#	Artifact	Description
A2.1	**API Documentation**	Application Programming Interface (API) documentation expresses how external software interacts with a game or its platform.
A2.2	**Build, Game Build**	Compiled version of a game. A build may have different purpose depending on the phase of the development. Notable builds are Alpha, Beta, and Release.
A2.3	**Build notes**	Build notes details the changes made in a build compared to previous build.
A2.4	**Source Code, Code, Script**	Source code is written instructions in human-readable form which will compile into a build. Code refers to machine-readable instructions compiled from source code.
A2.5	**Executive Review**	Conveys information about state of development between development team and upper management.
A2.6	**Prototypes**	Early model to acting as proof of concept or testing game feature. Different types of prototypes are paper, build, and technical.
A2.7	**Technical Design Document**	Outlines how the design should be implemented.

Engineering artifacts are related to the technical implementation of a game. The artifacts are presented in Table 3.2. At the start of development, prototypes are created. Prototype can be a physical paper prototype or build of a game. Technical prototypes can be used to test a game on a dedicated hardware. Game's functional

features are programmed into the game. Programming artifacts are source code and code. Source code is written instructions in human-readable language that are compiled into machine-readable instructions, code. Additional documentation regarding a game may be needed. Technical design document forms a basis for technical implementation of a game. Application programming interface (API) documentation expresses how external software is meant to interact with a game. A build is compiled version of a game. Different version of the game can be compiled into different builds depending on the purpose of the build. Alpha build of a game is early playable build to test game elements as a whole. Beta build is more advanced version of alpha build. The beta build is close to a finished version of the game. Beta build can be used to test the game with playtesters. When a game is ready to release, a gold master build is done. A gold master build is then copied onto physical media, a DVD for example. Gold master build may mean the same as release build, but release build is any build released to the users, including future updates. Patches are partial updates to a game that will address broken features, bugs. Each build and patch has build notes related to them which will list changes made compared to previous build. To convey progress information to the management, development teams will occasionally create executive review documents that communicate the progress made. [20]

3.1.3 Quality Assurance and Control

Quality assurance (QA) and Quality Control (QC) artifacts support developers to create smoother game experience for players. These artifacts are focused on fixing issues and improving performance in a game. QA and QC artifacts are presented in Table 3.3. Benchmarking and performance tools are both measuring performance with clear distinction. Benchmarking tools are meant measure the limit of hardware performance and performance tools are meant to measure performance of a software.

Table 3.3: Quality Assurance and Control artifacts adapted from the taxonomies [20], [21]

A#	Artifact	Description
A3.1	**Benchmarking Tool**	Tool intended to measure hardware performance in a game.
A3.2	**Bug Report**	A report of an issue in a game. Bug report will describe intended and actual outcome in a feature.
A3.3	**Performance Tool, Telemetry**	Tool intended to measure performance of software.
A3.4	**Playtest Materials, User Test**	Materials created for and during playtesting a game. Testing can include notes from testing, questionnaire for player, and comments from testing.
A3.5	**Test Artifacts**	Any artifacts related to testing, including different types of tests and test documentation artifacts.
A3.6	**Test Automation Artifacts**	Any artifact related to test automation, including test automation scripts and data.

Playtests aim to identify problems in a game from gameplay. Notes are made during playtest and players may fill questionnaire and give comments about the gameplay. Materials created for and from playtest are called playtest materials. Bug reports may be created from playtests too. Bug reports describe unexpected behavior of a game feature versus the expected behavior of it. Bug reports are then given to development teams who will decide what the appropriate course of action is. [20]

Notably, the taxonomy by Lee et al. are missing testing and test automation artifacts. Therefore, the taxonomy is augmented by adding artifacts A3.5 Test artifacts and A3.6 Test automation artifacts. These artifacts are adapted from Zabardast et al. Asset Management Taxonomy. Artifact A3.5 includes assets that are related to testing overall, testing assets and documentation for example. Artifact A3.6 includes assets that are related to test automation, test automation scripts and test automation data, for example. [21]

3.1.4 Art

Table 3.4: Art artifacts adapted from the taxonomy [20]

A#	Artifact	Description
A4.1	**Animatic, Ripomatic, Storyboard**	Art artifacts that help artists to plan and create scenes, feels and animations.
A4.2	**Art Asset**	Art piece that is used in a game, for example 3D models.
A4.3	**Art Notes**	Art notes contain information about specific art assets, for example context of an asset.
A4.4	**Art Bible**	A detailed reference guide for artists used to create consistent appearance for a game.
A4.5	**Concept Art**	Visualized concepts of game elements, levels or characters, as sketches or artworks.
A4.6	**Representative Art**	Artwork used in the distribution package of the game.
A4.7	**Typography Document**	Description of design and style of the typefaces.

The art subsection includes any visual art and related documents created during development. The artifacts are presented in the Table 3.4 Art artifacts are not limited to art present in a game itself but also includes art distributed with a game. Art documents created during development are art bible (also known as visual style guide), art notes, and typography document. Art bible is a reference guide to artists to keep appearance of the game consistent. Art assets are finished art pieces that are incorporated into a game. These can be 3D models of characters or items in a game. Art notes include information about related art asset. Typography document defines typeface to be used in a game. Storyboards are documents describing progression of a scene in a game. Animatic are further detailed storyboards that include animations. They are used to convey information that storyboards cannot present by itself, animations and changes in a scene for example. Ripomatics are short scenes of

other, existing media that presents desired mood and feel for a game. Finished art pieces can be external to the game in form of a logo or representative art. Representative art includes images and artwork used in the distribution package the game is released in. [20]

3.1.5 Audio

Table 3.5: Audio artifacts adapted from the taxonomy [20]

A#	Artifact	Description
A5.1	**Audio Asset**	Any and all audio instances used in a game. Includes audio effects, music, and dialogue.
A5.2	**Score**	Written form of audio asset, especially written music.
A5.3	**Voice Acting Artifact**	Information regarding voice actors/actresses and their performances.

Audio artifacts are similar to art artifacts, but they are mostly missing documents that are used to help with the audio design. The audio artifacts are presented in the Table 3.5. Audio assets are essentially any audio effects, music or tracks used in the game. Separate from the assets are written forms of music known as score and voice acting artifacts which includes everything related to voice acting. Voice acting artifacts contain information of actors/actresses, session takes and voice tests. [20]

3.1.6 Writing

Linguistic style, narrative, and setting of a game is portrayed in writing artifacts. The writing artifacts are presented in the Table 3.6. Story bible includes all the necessary information about a game's narrative. It includes lore elements and character traits which will be further detailed in narrative design document. Narrative design document is more detailed document depicting how a game's story elements

Table 3.6: Writing artifacts adapted from the taxonomy [20]

A#	Artifact	Description
A6.1	**Linguistic Style Guide**	A reference guide for script authors on textual and linguistic structure. Includes dialect and slang used.
A6.2	**Localization Document**	Information on translations from original language to other languages.
A6.3	**Setting document**	Description of the environment and setting the game takes place in.
A6.4	**Script (Narrative)**	Story of the game in written form for characters, including dialogue.
A6.5	**Story Bible**	A reference guide about game's narrative, including background information of characters.
A6.6	**Narrative Design Document**	More detailed description of progression of story elements, like plot.

progress. Script (narrative) is similar to movie scripts. It describes, story and dialogue of each character. Linguistic style guide dictates how the character will express what they are saying, including dialect and slang. Setting document describes the environment in which a game takes place. Translation from a game's original language to other languages is defined in localization document. [20]

3.1.7 Production

Production artifacts are any artifacts used for logistical management of game development. Production artifacts are listed in Table 3.7. Every asset used in making of a game is stored in a asset archive. Each type of asset will have their own asset archive, for example art or audio asset archive. Budget keeps track of development costs and how much resources are allocated to the game. To measure how the development of a game is progressing, a burndown chart is used. It shows what tasks are completed and how much time they took. Use of focus groups guides development of game features. The documents created from focus groups are called focus group

Table 3.7: Production artifacts adapted from the taxonomy [20]

A#	Artifact	Description
A7.1	**Asset Archive, Inventory**	Collection of resources for each type used in game development, for example art asset archive.
A7.2	**Budget (Game)**	Finances and resources for development of a game.
A7.3	**Burndown Chart (Agile)**	Measurement of progress in development of a game. Lists what features or modules are implemented.
A7.4	**Focus Group Materials**	Documentation regarding how focus groups view a game to guide game development.
A7.5	**Pitch Document**	Information document used to pitch, sell, a game to stakeholders.
A7.6	**Postmortem**	Reflection on development of a game, what went wrong and right, whether or not the game was released.
A7.7	**Schedule, Roadmap**	Schedule of development progression on a game.

materials. Pitching document is used to sell a game to its stakeholders, convincing them to support the game. Reflection on past development to improve future development is important. Development teams can create a postmortem document to go over what went well and what went wrong during development. Postmortems can be created after development milestones or after release of a game. Milestones and deadlines are formed into a schedule to keep track of development. [20]

3.2 Game Development Characteristics Contributing to Technical Debt

Even with increasing amounts of research in game development, technical debt is minimally researched in the field. The scope of related research in the topic is limited to two articles: ”Living With Technical Debt—A Perspective From the Video Game Industry” by Borowa et al. and ”Technical debt (TD) through the lens of Twitter:

A survey" by Alfayez et al. [22], [23]. Borowa et al. conducts their research in two steps: first step was to conduct a survey about technical debt with 12 participants from video game industry with control group consisting of 14 traditional software developers. The goal of the survey was to find characteristics that are specific to video game development by comparing deviation of median answers of both groups. Topics that were found to be video game development specific were used to form the questions for the next step, semi-structured interviews. Six participants from game development field were interviewed which yielded several characteristics contributing to technical debt in video game development. [22]

Alfayez et al. article studies interest in TD and how TD is addressed on X (formerly known as Twitter). Main focus of the article is not on game development and topics of the tweets are discussed to a limited extent. However, the article brings up interesting points related to game development which are worth exploring. These topics are crunch time and monetization over solving existing technical debt. [23]

Characteristics yielded from both papers will be discussed in this section. Each characteristic is first briefly explored in the context of game development to introduce how the characteristic is related to game development. After the introduction it is then discussed how these characteristic affect technical debt and what technical debt types can be considered to be related. At last, characteristic are linked to artifacts presented in Section 3.1. Any artifact that is mentioned in one of the two articles are considered to be strongly associated with the characteristics. If any artifacts can be deduced to be hypothetically linked to a characteristic based on the descriptions of characteristics and artifacts, they are then considered to be weakly associated.

3.2.1 Requirements, Change, and Planning

Requirements for video games are hard to establish. Non-functional requirements such as the "fun-factor" drive video game development as video games are intended

to produce emotional responses and are what differentiates games from each other. [22], [24]. First design of a game is usually just core features and concept art. These core features are developed into artifacts called prototypes (A2.6) and tested to find features which are well received. The most fit features are then incorporated into a game. Kasurinen et al. calls this approach *test and tune*. [24], [25]

Features are open for change throughout the development as they are tested and tuned. The features and the non-functional requirements set for them are evaluated through user testing. Feedback from testing is major part what shapes the game as the goal of the software is to arouse emotions. If developed feature is not contributing to this goal, a change is in order. [22], [24], [25]

As game projects may change at any point of the development, estimating, planning, and organizing development projects are often difficult. Vague planning leads to time pressure on the developers which also affects project's technical debt. [22]

In the interview conducted by Borowa et al., developers were found to be hesitant to put much effort into prototypes as they are susceptible to be cut from the final product. If the feature is decided to be included, any technical debt becomes part of the product if not paid off before hand. Decision to repay accrued debt in the feature is made when feature is decided to be included in a game. [22]

By reviewing different technical debt types introduced in Table 2.1, three potential debt types can be identified that could arise from description of requirements, change, and planning. The nature of constant change of game development brings the possibility of design and architecture debts. Constant changes all the way through the project may cause design principles to be overlooked. Also, it would be harder to adhere to good software architecture. Little evidence to these types of debt is given by Borowa et al. when developer is quoted to be "refactoring in secret because he cannot convince his superiors that paying off the debt is important." [22]. Developers ignoring quality most likely accumulates code debt too. [22]

Artifact A2.6 is the only clearly linked and mentioned artifact that is related to technical debt in this subsection. Prototypes are mentioned by Borowa et al. as part of feature development. [22] Artifacts that are part of prototypes can also be considered to potentially have debt. As the prototypes should be user testable, they are implemented as paper or build prototype. In the case of prototype build, any possible debt could be located in the code (A2.4) and any other artifacts that are part of a prototype build. It seems that changes can introduce debt in artifacts related to features based on the artifact descriptions. If feature goes through a change during development, the related design documents should also be updated. If the design documents are not updated, documentation debt is introduced in them. Artifacts A1.4, and A2.7 are prone to change when game features change as they contain descriptions of feature's design, and technical implementation.

3.2.2 Sub-par Testing Methods

Testing methods in game development is shown to be mostly manual testing. Borowa et al. questionnaire's results show that unit test coverage is just 15% compared to control group's 60%. The reason for focusing on manual testing is the focus on game projects' non-functional requirements and unacceptable raise of costs in feature prototyping. [22] According to Politowski et al., traditional testing strategies, like Test Driven Development (TDD), are found to be unfit in their current state for game development. Automated testing may increase cost of development, and requires knowledge and skills for successful implementation from developers. Instead, testing is organized as manual game-play testing by game testers. Game testers give feedback to developers whether the features of the game are enjoyable and fun, and bug reports are created based on bugs found. [26] Interviewees in Borowa et al. article express that automated testing could be utilized more, but it is not because of the current approach to testing in the industry. [22]

Despite the emphasis on manual testing and lack of automated tests, the industry is still aware of the importance of testing. Game testers' feedback affects the quality of the software and may eliminate certain types of technical debt from the project. [22] However, testing is often organized without proper planning which also affects test coverage. [26] Lack of test planning, insufficient test coverage, and minimal automated testing potentially introduces test debt and its sub-type, test automation debt, to the project.

None of game development artifacts introduced in Section 3.1 are directly mentioned to be related to technical debt in this subsection. Possible test debt due to lack of test planning is set in artifacts A3.4 and A3.5, and lack of test automation is tied to A3.6. If developers do not plan testing well enough, materials that guide testers to test comprehensively are lacking, leading to test debt in artifact A3.4.

3.2.3 Lack of Documentation and Methodologies

Petrillo et al. found that many game development projects are lacking in documentation. Postmortems examined in their research indicate that documentation was inadequate in design documents but also in finished sections of the product. Poor documentation of the project is further affected by communication problems. It was also found that communication between technical and creative teams was difficult. [27] Borowa et al. confirm the same issues in communication and documentation in game development projects. Communication and documentation issues will lead to phenomenon called knowledge vaporization, which will culminate when developer decides to leave the company without sharing the knowledge. [22] Interviewees also claimed that "nobody actually knows how to make games properly". Inadequate understanding, and missing standardization of game development methodologies can be attributed to low amount of specific research in the field. [18], [22]

Poor documentation and lack of knowledge sharing could lead to more technical

debt. Lack of up-to-date documentation itself is already counted as documentation debt, but inadequate knowledge about project, its state, and artifacts may lead to more technical debt with further development. [22] Challenges in knowledge sharing may be harder to counter in game development with missing standardization of methodologies. A new developer must learn working methods and practices of the workplace in addition to learning the state of the project without proper documentation. Accumulation of other debt types may be worsened by compounding challenges in knowledge sharing and missing methodologies.

This subsection does not mentioned any specific artifacts to have technical debt. Documentation debt created from inadequate documentation could reside in any documentation artifact created during development. Following artifacts could contain technical debt based on their description:

- **Design artifacts** A1.3, A1.4
- **Engineering artifacts** A2.1, A2.3, A2.7
- **QA artifacts** A3.2, A3.4
- **Art artifacts** A4.3, A4.4, A4.7
- **Audio artifacts** A5.3
- **Writing artifacts** A6.2, A6.3, A6.5, A6.6
- **Production artifacts** A7.4, A7.7

These artifacts include every documentation related artifact that could have technical debt. Creative documentation related artifacts, like art, audio, and writing artifacts, are also included.

3.2.4 Monetization over Software Quality

Borowa et al. claims game developers may be deliberately accumulating technical debt based on the current and predicted future sales of the product. Authors state that since video game players are insensitive to the quality of the game, developers may deliberately take on additional technical debt, if the game is known to be profitable. [22] Little evidence is given to support this statement or extent of it in the article, but it still valuable to acknowledge that developers may introduce additional debt to the project if profitability goals of the game are met.

Monetization of video games has also been discussed on X (formerly known as Twitter). According to Alfayez et al., users complain that monetization of the video games are prioritized to the detriment of software quality and technical debt management. [23] The two articles about monetization has each a tiny distinction between them: while Borowa et al. suggests that monetization strategy may allow developers to accumulate debt intentionally, Alfayez et al. say that "industry games are more catered toward monetizing their games at the expense of solving their TD". [23] The difference is small, but latter may not intentionally accumulate more debt, just the management of technical debt may be neglected.

Neither of the articles discuss any particular type of debt, but rather give insight to technical debt management in the industry. Developers may intentionally take additional technical debt. Prioritizing monetization over technical debt repayment is neglecting management of technical debt, but may not be intentionally accumulate more debt. It is not clarified in either case whether project's technical debt is subjected to any debt management activities or strategies.

Prioritizing financial gain instead of software quality and technical debt management does not affect in any specific artifacts, but instead affects the amount of debt overall in the project.

3.2.5 Dormant Debt

Old technical debt instances may affect future development effort on projects. Borowa et al. reported that one interviewee was part of a remastering project. Remastering game means to improving old game to bring it to modern platforms. The said interviewee reported that key artifacts were often missing and the company may have to develop their own compiler to compile the legacy code involved in the project. [22]

Technical debt from previous development effort is carried over to any future development on the same project. Effects of technical debt is being emphasized as artifacts are incomplete. Hypothetically, effects of documentation debt is even more significant as undocumented knowledge is lost with developers of the original development effort. Development team needing to develop a compiler to run legacy code implies that deprecation of development environment and old versions of programming languages may introduce code debt.

No specific artifacts are mentioned to be strongly associated with dormant debt. Based on the literature, artifact A2.4 can be deduced to be associated with remastering projects and without concrete proof, it should be considered as weakly associated.

3.2.6 Crunch Time

Crunch or crunch time is term used to describe overtime work periods in the game development industry. During the period which can last weeks, or in some cases months, developers are overloaded with work trying to keep up or catch up to the schedule. [28] Three main reasons have been identified for crunch: issues with scope, feature creep, and deadlines, deadlines being the most common reason. [28] Edholm et al. found four different types of crunch: delusional, continuous, mini, and final crunch. The effects of the crunch on the project varied between the types of the crunches. In continuous crunch, where crunch was present through majority of the

project, developers were found to implement non-requested features including those that were not planned in advance. Continuous crunch also affected software quality. Developers may not have enough time to reflect on the issues presented and in some cases it had led to shortcuts. Mini crunches, short periods of crunch during the project, were found to improve the software quality and preserve the original features planned. Final crunch which happens at the end of the project, had mixed results, either improving or worsening software quality depending on the state of the project prior to the crunch. There was not enough evidence on the effects of delusional crunch. [28]

Crunch's effects on technical debt seems to depend on the type of crunch in question. Based on the Edholm et al. description of mini-crunch, the increasing software quality could mean that technical debt is also paid off. However, continuous and final crunch may have opposite effects on technical debt. Continuous crunch has detrimental effects on the quality of the software and it could mean larger amount of technical debt. Authors Edholm et al. mention that "In a few cases there have been shortcuts made in order to save time, which in the end has led to more bugs" which would fit to the description of technical debt well. [28] The same applies for final crunch in some cases. Posts on X studied by Alfayez et al. also complain about the phenomenon adding technical debt. According to the posts, crunch does not address new or existing technical debt, but is used to create new features and finish existing ones. [23] Based on the literature, it is difficult to say whether crunch accumulates any specific type of debt over others, but it seems to create more technical debt. There is no evidence of additional debt being added to any specific artifacts.

3.3 Summary

In this chapter, game development artifacts are adapted and augmented from a taxonomy by Lee et al. to present most relevant artifacts for the thesis. The total

of 41 artifacts are categorized in seven subsections: design, engineering, quality assurance and control, art, audio, writing, and production.

Six game development characteristics are presented and explored how they are known to affect technical debt in game development. Each characteristic is further examined to reveal any possible ties to technical debt types and game development artifacts. A summary of these relationships is presented in Table 3.8. Technical debt types are represented with corresponding identifier determined in Table 2.1. Artifacts with strong association to a characteristic is written in **bold**.

Table 3.8: Summary of characteristics, technical debt types, and artifacts

Characteristic	**Technical Debt Types**	**Artifacts**
Requirements, Change, And Planning	T1, T2, T3	A1.4, A2.4, **A2.6**, A2.7
Sub-Par Testing Methods	T4, T5	A3.4, A3.5, A3.6
Lack of Documentation and Methodologies	T6	A1.3, A1.4, A2.1, A2.3, A2.7, A3.2, A3.4, A4.3, A4.4, A4.7, A5.3, A6.2, A6.3, A6.5, A6.6
Monetization over Software Quality	-	-
Dormant Debt	T2	A2.4
Crunch Time	-	-

4 Researching Technical Debt in Game Development

To achieve the goal of the thesis, semi-structured interviews are conducted to understand how technical debt is accumulated, in which artifacts it resides, and how it is managed in game development. In this chapter, the interview structure, how participants were selected, and the results of the interview are explored.

4.1 Interview Structure

The interview is divided into five main sections: introduction, overview of technical debt, debt in game development artifacts, accumulation of technical debt, and technical debt management. Each section was formed to support answering each of the three RQs formed in Section 1.2. Interviewees were asked in total nine questions, each labeled Q#. Complete list of questions can be found in Appendix A. Prior to the interview, interviewees were asked demographic questions Q1, Q2, and Q3 to determine their experience in the field.

First section of the interview, introduction, introduces the author of thesis, and the goal of the thesis to the interviewee. Interviewee is at the same time informed how data is presented in thesis and their right to withdraw from the interview.

Next an overview of technical debt is presented to the interviewee. A presentation is shown to interviewee with definition of technical debt used in thesis, definition

of technical debt components presented in Section 2.1, and focused technical debt types presented in Section 2.2. An overview of technical debt is presented to ensure interviewee has the same understanding of the concept as used in the thesis. After the overview, Q4 was used to establish interviewee's familiarity with technical debt and how often they encounter it.

In the next section of the interview, Debt in Game Development Artifacts, interviewees were introduced to the taxonomy of game development artifacts used in the thesis. Interviewees were also cautioned that the taxonomy of artifacts may be incomplete, or the definitions of artifacts may be different due to lack of standardization in the industry as discovered in Section 3.1. Artifacts were explored a section at a time with presentation including names and descriptions of the artifacts shared to the interviewee. At each section of artifacts, the same Question Q5 were asked: in which of these artifacts have you encountered technical debt? After last of the artifacts were presented, interviewees were asked if any other prominent artifacts were missing from the taxonomy and whether they had any other thoughts or ideas regarding technical debt and game development artifacts. Answers gathered from this sections will provide answer for RQ2.

The interview continues with discussion about accumulation of technical debt. During the interviews, some of the reasons for debt accumulation was already described in previous section when artifacts were explored. Most of the interviewees contemplated the reasons why some artifacts accumulated technical debt at the same time as they explained which artifacts could contain technical debt. Technical debt accumulation in game development were further assessed with one Question Q6: What factors affect accumulation of technical debt? As with other sections, at the end interviewees were asked if they had any additional thoughts or ideas regarding accumulation of technical debt. This section was designed to support answering RQ1.

Last section of the interview covers technical debt management in game development. From total of the nine questions, the last three focused in this part: Q7, Q8, and Q9. These questions are formed from technical debt management activities discussed in Section 2.3.1 The goal of these questions is to discover how technical debt is managed and whether technical debt management methods are utilized in game development. Reactive technical debt management is not the only point of interest, but also proactive technical debt management. Interviewees were also asked what methods are used to prevent accumulation of technical debt, and how possible technical debt instances are documented. The answers to the these issues provide answer for RQ3.

The interview was specifically arranged so that artifacts were presented and reviewed first before proceeding to the remaining interview questions. The goal of this arrangement is to let interviewees think more about technical debt from different viewpoints in hope that more extensive answers could be provided to later questions.

4.2 Recruiting Interview Participants

Table 4.1: List of interviewees

I#	Occupation	Experience, software development (years)	Experience, game development (years)
I1	Lead Developer, Senior Programmer	11	11
I2	Narrative designer, Game designer	10	10
I3	Senior Software Developer	10	5
I4	Game programmer	20	16
I5	Game programmer	14	14

Scheduling and conducting interviews proved to be longer and more difficult process than anticipated. The participants for interviews were sought exclusively from

Finnish game development industry. The process started with contacting first participants in February 2026 and conducting first three interviews the same month. Unsatisfied with the number of interviews held, the author of the thesis continued sending interview invitations to game development companies via email, game developers through social media, and posting the invitation in game development community on a social platform. Three university educators were asked for possible contacts in the industry. Overall, the author of the thesis contacted 23 companies and 6 individuals, with 6 responses to the invitation overall and 5 developers proceeding to the interviews. The interview process ended after two months, in April 2026. The list of participants with occupation, experience in software development, and experience in game development is presented in the Table 4.1. Participants I1, I2, and I3 are working in a smaller, independent game developer Company X, and participants I4 and I5 are working in a larger, established game development Company Y.

All of the interviews were held in Finnish. Introduction slides of technical debt and artifact taxonomy were translated to Finnish for the interviewees, but artifacts and their descriptions remained in English. Translating artifacts to Finnish was deemed too difficult as definite artifact terms are not established in Finnish. Quoted responses given by the participants were translated to English for the thesis.

4.3 Results on Overview of Technical Debt

After presenting an overview of technical debt, interviewees were asked the question Q4. All of the interviewees answered the question by stating that they encounter technical debt often in game development projects. Interviewees that mentioned frequency of encountering technical debt explained that it is weekly, if not daily, occurrence in their job. Some interviewees stated that how often they encounter technical debt depends on a project. Age of a game development project affects the

amount of debt in it. Technical debt may be a daily occurrence with older projects, but a new project may not have any debt, especially when starting with a clean slate. However, new project may already have technical debt, as I4 stated:

> Yeah, in new games, if there is any technical debt, it often has to do with the fact that you still use some older components that were developed earlier. For example, you might have a game engine that was originally made for a somewhat different kind of game, and that's where the debt comes from. (**I4**)

Overall, interviewees were very familiar with technical debt, it being a common occurrence in game development projects.

4.4 Results on Debt in Game Development Artifacts

In the following subsections, technical debt's relevancy to game development artifacts are explored based on the answers of the interviewees. The answers are formed into a table in each subsection and they are coded in the following way:

- **TD** The artifact has technical debt
- **md** The artifact may have debt
- **CD** The artifact causes debt
- **ND** The artifact does not have debt or debt is not applicable to this artifact
- **R** The artifact is not used by the interviewee, interviewee is not familiar with the artifact or the artifact is not relevant to the interviewee

The artifacts are coded this way to differentiate artifacts that have the most potential to have technical debt, artifacts that may not have debt but cause it, artifacts that usually does not have debt or debt is not applicable, and artifacts that interviewee is unable to comment on whether they have debt or not. Artifact names and descriptions are defined in Section 3.1.

4.4.1 Design Artifacts

Table 4.2: Identified debt in design artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A1.1	ND	ND	R	ND	ND
A1.2	ND	TD	R	R	ND
A1.3	TD	md	R	R	R
A1.4	TD	TD	TD	R	TD
A1.5	R	R	R	R	R

All of the interviewees stated that documentation overall is often outdated or completely missing. Design artifacts are almost exclusively living documentation artifacts, which describe the design of the game. The same issues with documentation apply to design artifacts. Lack of resources and time allocated to documentation upkeep allows design artifacts, which may not be easily updated, to fall behind of the actual design. Some interviewees stated that sometimes only the ideas documented in Artifact A1.1 is available of the design. The contents of Table 4.2 are examined in the following subsection.

Interviewees explained that usually documentation is appropriately done to A1.1. None of the interviewees saw technical debt applicable to A1.1 as the nature of the artifact is to lightly document ideas brought up during brainstorming as explained by I5:

> My experience has mostly been that at the point where you start writing something down, this can also just be a really lightweight, PowerPoint-style presentation with a few slides. In that case, I wouldn't really see a lot of technical debt, because at least in my experience it has been very much a kind of throwaway thing - it's done once, for example to present a concept or something like that, and then it doesn't really get further developed all that much. (**I5**)

These ideas are further developed into actual design documents, which may have technical debt in them. A1.2 is used to refine ideas discovered during brainstorming. Interviewees I1 and I5 consider A1.2 to rarely have debt for different reasons. According to I1, A1.2 is used to convey ideas between developers, which would require the artifact to be up-to-date and free of technical debt. I5's reason for it being debt free is the nature of the artifact being similar to A1.1, not relevant whether the artifact has debt or not. On the other hand, I2 mentioned that A1.2 is an artifact that could have a lot of technical debt as it may deprecate quickly. I2 added that it is more important to keep the artifact up-to-date in early stages of production to share ideas between designers when no concrete implementation exists, than in later stages when the actual implementation of it exists.

A1.3 is twice mentioned to have technical debt, but little knowledge how much debt it may have is known. Interviewee I2 believes this artifact could have a lot of technical debt, but has never had the opportunity to be part of creating one, only using the artifact. I2 explained that since the artifact should communicate design choices, even to outsourced resources, it is important to keep up-to-date. I1 mentioned that the artifact together with A1.4 are often inadequately documented.

Every interviewee who has experience with A1.4 indicated that the artifact is the most likely to have technical debt in design artifacts. In the interviews the artifact was often described to be easily outdated and requiring constant effort. A number of

interviewees claimed that the artifact may have so much debt from being outdated that it may prove useless, as I5 explained:

> My experience has quite often been that, even if there is some kind of game design document to begin with, at some point it starts to fall apart so much that it's basically no longer useful. (**I5**)

The last artifact, A1.5, is the only design artifact that none of the interviewees had any experience creating it, and as such, it is unknown whether technical debt applies to it, and in what quantity.

4.4.2 Engineering Artifacts

Table 4.3: Identified debt in engineering artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A2.1	ND	R	ND	TD	TD
A2.2	ND	md	ND	ND	ND
A2.3	md	md	md	ND	md
A2.4	TD	R	TD	TD	TD
A2.5	R	R	R	R	ND
A2.6	ND	ND/CD*	ND	ND/CD*	ND/CD*
A2.7	TD	TD	TD	TD	ND

From the point-of-view of traditional software engineering, the engineering artifacts might be the most interesting section. Interviewee's answers are presented in Table 4.3. Two artifacts are easily identified as the ones with most technical debt: A2.4, and A2.7.

According to the interviewees, the amount of technical debt in Artifact A2.1 depends on how APIs (Application Programming Interface) are used and on the size of the development team. If APIs are intended for use within a development

team or the API has narrowly defined use, the artifact may be completely missing or otherwise lacking as the documentation is not seen as important. Interviewee I5 noted that the importance of A2.1 grows as team sizes grow:

> Based on my experience, this [Artifact A2.1] is usually in pretty poor shape when it is maintained internally within the company. In a way, the bigger teams you have and the more users there are, the more effort tends to be put into it. (**I5**)

I5 continued that if some feature with an API has its own dedicated team creating it, A2.1 is usually well put together contrary to a feature that is created to be used within a team. Interviewees found that A2.2 is mostly free of technical debt. I2 described that only they know how the builds are made as they have not explained it to anyone else. Issues could arise if someone else from their team should build a game version. I2 states that the way builds are done is not documented, introducing technical debt. Most of the interviewees declared that how much different builds are documented in A2.3 depends on the build. Developer builds, which are used by the developers to test a game, often lack documentation about changes done between builds. Some interviewees reported that when a new version of a game is released to the public, a change-log containing the most notable modifications is put together based on the version control history and pull requests. Even then the artifact created may not contain every change made between the builds. I1 noted that most important changes are in the change-log:

> These patch notes and the like do not necessarily always actually include everything that has been changed; it is more like that the things we see as worth mentioning are the ones that maybe end up being there. (**I1**)

I4 was the only who explained that A2.3 is often well made in bigger teams, as other teams' work may be more challenging as the result of doing otherwise. The

most prevalent artifact for technical debt was A2.4, which was expected. Every interviewee familiar with A2.4 said that the artifact is the most obvious and common place for technical debt. Technical debt in the artifact often affects how extendable it is. Interviewees reported that they have had to either refactor entire features to make them easier to implement extensions to it or in some cases, implement a completely new system parallel to old one which increases the complexity of the system, and necessitates the upkeep of two different systems at the same time. I1 explained that features that are such deeply woven into the project that refactoring the feature into more flexible one may take too long, including retesting it, and it is faster to implement a new one. I4 expressed that to save time, a feature may be intentionally designed and implemented sub-optimally, betting on that the feature won't be extended further in the future:

> When a system is originally made, a conscious risk is taken, in a way - that okay, we'll do this like this now. We know that this won't necessarily scale in the future, but on the other hand we also don't know whether it will need to scale in the future. (**I4**)

I1 reported that important features they are expecting to be extended in future will have more effort and time put into them so that possible future problems can be avoided. I4 continued that the opposite is also possible, starting to create a feature that is flexible, but shortcuts must be made during the implementation due to time constraints. Some interviewees also mentioned that debt in the artifact may be also caused by insufficient knowledge about how to create the optimal solutions.

Most of the interviewees had no experience with Artifact A2.5 except I5 who said that the artifact itself does not contain technical debt but technical debt and the resources needed for repaying it might be challenging to communicate to the management.

Every interviewee experienced with A2.6 agreed that the artifact itself is a tool

used to avoid technical debt and may purposefully have technical debt in it. Interviewees explained that the artifact should be only be used to demonstrate and test possible future game features, and when used correctly, it should be discarded when it is not needed anymore. Instead, approved, prototyped features should be built from ground up with properly implemented parts. However, interviewees reported that from time to time, the artifact is not used correctly and it is not discarded due to the time pressure. Instead, the new feature is built on top of the artifact, retaining any technical debt already in it. In this way, any debt already in the artifact will move on to the project.

The last artifact on the list, A2.7, is often outdated or completely missing. According to the interviewees, technical design is rarely documented in a comprehensive manner. I5 said that developers may create some design documentation how a game feature should be implemented prior to the implementation for themselves, but a complete artifact about the technical solution for whole system is never created. A number of interviewees also noted that game development teams usually lack a dedicated person to create a software architectural design as the reason for the missing artifact. Even if some documentation has been created, it will often deprecate quickly if it is not kept up-to-date. I1 explained that after some point the amount of documentation debt might be so massive, that it is no longer worth to repay:

> Now there's so much code. When it's been written for four years, or three years, whatever it is, at this point it's kind of impossible to start documenting it. Or it's not impossible, but you could spend two weeks on it, and then it just does not get done.

4.4.3 Quality Assurance and Control Artifacts

The quality assurance and control section contains artifacts that are challenging to evaluate with certainty in terms of technical debt due to the limited use and nature

Table 4.4: Identified debt in quality assurance and control artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A3.1	R	R	R	md	R
A3.2	R	md	TD	TD	md
A3.3	R	R	md	md	ND
A3.4	R	R	R	R	ND
A3.5	R	R	R	R	R
A3.6	R	R	md	TD	R

of testing in the industry. Interviewees I1, I2, and I3 explained that Company X does not have any dedicated team for testing, and most of the testing is conducted by the developers in form of playing and testing the game. Overview of interviewees' answers are presented in Table 4.4.

Interviewees did not have enough experience with Artifact A3.1 to tell about technical debt in it with confidence. I4 reasoned that the artifact could be used in way that does not fit the purpose of it well, or it is not measuring the correct parameters as possible technical debt in the artifact.

Most of the interviewees reported that repository where A3.2 artifacts are stored often has redundant individual artifacts in them. Interviewees reported three different ways the A3.2 artifact and the repository could have technical debt: reports about bugs that are already fixed, bugs that are not relevant anymore because of a change in the system, and smaller, low priority bug reports staying the reporting environment for long periods of time. Developers may spend time unnecessarily fixing bugs based on the artifact if the bug is already fixed in another artifact. Bug fixing is prioritized based on the impact of a bug, leaving smaller bug in the repository for a longer period of time, even years, as I5 explained:

> Often with bug reports you get situations where there are bugs that end up lingering there for years, because they're never at a high enough

> priority for anything actually to be done about them. And then when you move years forward, they start to accumulate there, and the bug system itself is full of all kinds of stuff that nobody really knows anymore - or that nobody is really even interested in anymore - what's actually in there. The highest-priority bugs of course get fixed, but there's so much junk there that nobody has even checked in a few years whether it's still a problem at all. (**I5**)

These smaller, low prioritized artifacts may linger in the repository so long that a system change will make the artifact redundant.

According to the interviewees, Artifact A3.3 rarely contains technical debt. I3 stated that modern game engines, like Unity or Godot, already provide necessary performance and telemetry tools, but technical debt could be a problem if the game engine is developed by an in-house team. In that case, the developers would have to develop their own tools, and technical debt could be more significant aspect. I4 also noted that the artifact may become outdated for the desired use, and a new system might have to be developed on the side. At the same time, the implementation of the old system may still remain in the system, and the deprecated artifact may move on to new projects through code reuse. I5 said that they have never encountered anything that they would call technical debt in the artifact.

Interviewees I1, I2, and I3 did not have any experience with Artifact A3.4 as it is not used in Company X. Only interviewee I5 had experience with the artifact to say that the artifact does not have debt as usually old questionnaires are used and if it does not fit the purpose, a new one is made.

Artifact A3.5 was not familiar to any of the interviewees. Interviewees explained that most of the testing is unorganized, ad hoc testing which lacks testing strategies, plans, and reports. Developers usually test their own implemented solutions.

Similarly to A3.5, A3.6 is rarely used due to the challenge of automating tests

that would represent what players would do in a game. Interviewee I4 has experience with the A3.6 and stated that the test should be abstracted so that the implementation of tested feature does not matter in the testing. However, often developers may need to use test data that is fed into the system being tested. If changes are made to the format of data in the implementation, the artifact may become outdated as the test data may not be updated, causing technical debt. I5 said that the issue is caught when tests in A3.6 start to fail. I3 noted that the artifact is not usually used in game development, but may be used to test backend if a game has a separate backend, but even then the artifact is used to test the APIs.

4.4.4 Art Artifacts

Table 4.5: Identified debt in art artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A4.1	R	R	ND	R	R
A4.2	md	R	md	md	TD
A4.3	md	R	R	R	R
A4.4	R	R	R	R	R
A4.5	R	R	R	R	R
A4.6	R	R	R	R	R
A4.7	R	R	R	R	R

Due to role-wise homogeneous group of interviewees, most of the artifacts were left unexplored as the interviewees did not have any experience with them. However three artifacts were recognized in varying decrees: A4.1, A4.2, and A4.3. Overview of technical debt in art artifacts is shown in Table 4.5.

According to I3, A4.1 rarely has issues with technical debt. The artifact is usually properly created the first time and if any improvements are needed, they are iterated into a new version. Artifact A4.2 is most commonly used artifact by interviewees

and has the most potential to have technical debt. Artifact A4.2 may have multitude of different problems: placeholder art making into the game without replacement, neglecting prefab creation, outdated file format, and suboptimal creation of assets. I1 mentioned that if artist do not have time to create art assets when developer needs them, developers may use placeholder art made by themselves. Sometimes these placeholder assets remain in the game instead of being replaced with artist created assets, accumulating technical debt. In game development, prefabs are complex objects that are pre-configured into templates that can be then reused in the creation of a game. [29] Artifact A4.2 is commonly used in prefabs, and updating the artifact in the prefab would replicate the update to every instance of the prefab in the project. According to I1, if A4.2 is not configured as a prefab, every instance of the artifact must be updated by hand, creating debt into every instances of the same artifact in use. The format of Artifact A4.2 may deprecate when environment of the game development project develops. Interviewee I4 indicated that technical debt may develop into A4.2 if used technology progresses without corresponding changes in the artifact:

> In the assets themselves, there have been some situations where we have an asset that was made, say, years ago already, and then something new comes along, like for example a new compression format that some newer GPUs [Graphics Processing Unit] support and things like that. So you're like, okay, we need to convert this asset into a different format so that it saves, for example, memory. (**I4**)

At last, I5 explains that the process of creating A4.2 artifacts may not be optimal. Sub-optimal creation of art assets may rise from lack of knowledge in how to use said assets in the game engine. I5 gives an example of how technical debt may be introduced this way:

> These [art assets] weren't necessarily done wrong, but done in a very, let's

> say, inefficient way. And then you're in a situation where you suddenly already have, like, 50 characters' worth of assets made or something. And then you start to be in a situation where, okay, do we start remaking these or not. And that's maybe something that you'd see more in the art assets, or what I would count as technical debt in art assets. (**I5**)

I1's description of possible technical debt in Artifact A4.3 is connected to Artifact A4.2. If placeholder assets are used in a project, it should be well documented in Artifact A4.3. The lack of documentation in A4.3 about the use of placeholder assets is documentation debt.

4.4.5 Audio Artifacts

Table 4.6: Identified debt in audio artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A5.1	R	R	md	R	ND
A5.2	R	R	R	R	R
A5.3	R	R	R	R	R

Audio artifacts were similarly unknown to interviewees as art artifacts in the previous Section 4.5. Overview of interviewees' answers on audio artifacts is presented in Table 4.6. Interviewees rarely encountered Artifact A5.2, and Artifact A5.3 were not relevant to the projects interviewees have worked on. Interviewee I3 was the only interviewee to identify possible technical debt in A5.1. They stated that if multiple people participate in the creation of A5.1 artifacts, they may not be consistent. As for an example, I3 said that the artifacts may have different levels of volume, lacking a mastering stage in the process. Interviewee I5 said that they have not encountered any situations where A5.1 would have technical debt.

4.4.6 Writing Artifacts

Table 4.7: Identified debt in writing artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A6.1	R	TD	R	R	R
A6.2	TD	TD	R	R	ND
A6.3	R	TD	ND	R	R
A6.4	R	TD	TD	R	R
A6.5	TD	TD	R	R	R
A6.6	R	md	R	R	R

Most of the artifacts in the writing sections were not familiar to most of the interviewees. Interviewees' roles are mostly focused on software development, which would not have any writing related artifacts. Interviewees working in the Company X were somewhat familiar with some of the writing artifacts, as developers in smaller project tend to be more familiar with multiple domains. Interviewee I2 were most familiar with artifacts in this sections as they are the best fit role-wise. Overview of the writing artifacts are presented in Table 4.7. Some interviewees stated that lack of documentation in writing artifacts is often caused by inadequate resources allocated to writing artifacts. Instead of allocating resources to documenting writing, writing new narratives and progressing script was prioritized.

Artifact A6.1 was indentified by one interviewee to have technical debt. Interviewee I2 described that the artifact was completely missing, and no documentation about linguistic style exists. They described that any linguistic style created by them for the game is not documented anywhere.

As for the Artifact A6.2, I1 and I2 stated that the artifact does have debt at least in the projects they have worked on, but in the other hand I5 had never encountered anything that they would describe as technical debt. I1 and I2 argued that projects they have been working on would not have localization of any kind,

and if localization were added to the game, the first thing to do would be to make a document like Artifact A6.2.

Two differing statements about Artifact A6.3 were made as I2 stated that the artifact is not often updated and would have deprecated a bit, but I3's view of the artifact was that it is often important artifact to be up-to-date, therefore it would seldom have technical debt.

Interviewees I2 and I3 agreed that the next artifact, A6.4, could often contain technical debt. Interviewee I2 stated that it would be pointless and redundant information to have a separate documentation for the narrative, as it would be difficult to document the narrative script in whole. The narrative is also already readable in the implementation, so it would not be cost-effective to document it in Artifact A6.4 again afterwards. I3 pointed out that any change in artifacts tied to A6.4, like the design documentation, would introduce technical debt in to the artifact, until it has been updated to reflect current design.

Like the artifacts explored above, Artifact A6.5 is no different. I1 and I2 explained that any additional information about narrative and background information is not documented to the artifact, as it is completely missing. Intricate details of narrative choices are not documented separately, and the only written knowledge can be found in the implementation of the game. The same applies for the last Artifact A6.6. I2 explained that they have been the only person in the project to take part in writing, and they did not find it essential to keep all of the writing documentation artifacts up-to-date.

4.4.7 Production Artifacts

Last section of the development artifacts is production artifacts. The various artifacts in this section were somewhat familiar to the interviewees, but also contained artifacts that were not used in the projects interviewees have been part of. An

Table 4.8: Identified debt in production artifacts

A#	**I1**	**I2**	**I3**	**I4**	**I5**
A7.1	TD	TD	md	R	R
A7.2	R	R	CD	R	R
A7.3	R	R	R	R	ND
A7.4	R	R	R	R	ND
A7.5	R	R	R	R	ND
A7.6	R	R	ND	ND	ND
A7.7	TD	TD	CD	ND	ND

overview of the artifacts is presented in Table 4.8.

First artifact in this section, Artifact A7.1, could have technical debt according to the interviewees. Interviewee I1 stated that assets from the archive could be modified in the projects they are used in, but the new, modified versions might not be updated to the archive. This will then create multiple versions of them, and the asset in the archive is then deprecated. If the asset is then used in another project, developers need to seek out the modified version from a different project. I2 also explained that file hierarchy and naming conventions are often neglected, making the assets and their versions harder to find.

Most of the interviewees had no experience with Artifact A7.2, but I3 said that it often causes problems and technical debt in the project. They did not specify whether the artifact could have technical debt in itself. For the next three artifacts, A7.3, A7.4, and A7.5, only one interviewee, I5, had some experience with them. For A7.3, the interviewee described that the artifact does not have debt in it. Artifact A7.4 is often ad-hoc in nature, and the materials are not usually reused, so technical debt would not apply to it well. The same applies to the Artifact A7.5, the artifact may be used in the start of a project, but usually it is not needed the further the project progresses.

Technical debt does not apply well to Artifact A7.6. The artifact is a report that is written after the project concludes, which may vary how extensively and how well it is done, but interviewees did not associate technical debt to it.

The last artifact in this section, A7.7, divided the opinions of interviewees in half. I1 and I2 explained that the artifact is often outdated, as schedule lives constantly and it is difficult to estimate how much time could be allocated to each project. I4 and I5 stated that as the artifact is a living document, it does not make sense to make detailed plans, and changing schedule does not introduce debt, but incites the need to create a new version. I3 explained that difficulties estimating and creating the artifact is what causes technical debt in projects.

4.5 Results on Accumulation Of Technical Debt

This sections discusses results on technical debt accumulation in game development. In the interview structure, Question Q6 was dedicated as basis for discussion about what factors affect technical debt accumulation in game development. Most of the interviewees also discussed accumulation of technical debt during previous Section 4.3. When the interviewees explained which artifacts could contain technical debt, they also often discussed the reasons why debt is accumulated to the artifact.

Most common reason for technical debt stated by the interviewees was lack of resources. Limited resources drives developers to balance between the scope and the quality of the project. Developers compromise in quality of the project to keep the scope of the project large enough, introducing technical debt in to the project intentionally. The amount of debt introduced is larger if the scope of the project is too large, as developers may be hesitant to cut unnecessary features. According to the interviewees, lack of time and personnel drives developers to create minimum viable features that fill the requirements for the current need. The developed feature may not be created according to the best practices, and may not be easily extensible

without additional work as I1 and I4 explain:

> And I could imagine that it's generally common in small companies as well, that when there are limited resources, few people, and basically small budgets compared to what is being tried to achieve, it leads to a situation where often you first do what is the minimum requirement, and then with the hope that at some point it will be improved or expanded, but then it's common that if it is good enough for the final product, it might just stay like that ... because the other option is that that thing doesn't exist at all. So if the options are that it's done in the best possible way or that it's done well enough, then if doing it well enough isn't an option, then it doesn't get done at all. (**I1**)

> So instead, it might just be that the schedule kind of dictates that we can't do it that way, and we sort of keep piling technical debt on top of the old stuff. We end up doing things that, at the time, we already know will just create more debt, but it's just one of those things you can't really do anything about. (**I4**)

According to the interviewees, poorly implemented features may accumulate even more technical debt on top of the initial debt, if developers can not use enough time to repay the previous debt. Ideally prototype features should be created in a sense that they are going to be thrown away or rewritten if the feature is proceeding to be added to the game. However, in some cases prototype code must be used as a basis in development due to restrict schedule. I4 describes the additional debt accrued:

> Yeah, and often even if it is recognized that, okay, now we have technical debt here and ideally we should rewrite some system to get rid of that technical debt, instead it may be that the schedule simply dictates that we can't do that, and we continue accumulating technical debt on top of

> the old one - that we do something that we already know at that moment will just add more debt, but it's just something you can't do anything about. (**I4**)
>
> If this ends up becoming something more than just a prototype - if it takes off from there and gets developed further - then there can be situations where things have been done a bit with the idea that yeah, this will be rewritten, but then for some reason it turns out that we can't rewrite it after all. We have to use this prototype code, and then at a fairly early stage technical debt has already been created. (**I4**)

Lack of time and personnel does not only affect implementing game features, it also applies to documentation. Interviewees I1 and I2 explained that time pressure may lead to inadequate documentation when designing a game. Designers may prioritize creating new designs instead of documenting already existing or recently created designs. Creating new designs for the game advances the project, while documenting the design is not as beneficial. According to some interviewees, documentation is perceived as less important when the design has already been implemented to the game, or if the documentation would potentially have limited amount of users. Minimal documentation leads to future problems with technical debt. At the time of artifact creation, documentation may not have been important factor. Problems arise when a developer with detailed knowledge about a feature leaves the team, and the feature is further developed in the future. I2 explained that the absence of knowledge sharing hinders the development as complex implementations take time to understand without documentation.

Multidisciplinary nature of game development is emphasized when every team member does multiple different roles. According to I1, smaller teams may not be able to hire dedicated personnel for each role due to limited budget. As every team member can not be knowledgeable or trained in creation of artifacts that differ from

their usual work, they may introduce technical debt unintentionally.

Incomplete or changing design in itself is also a common factor adding to the technical debt. Interviewees stated that design of the feature may not be ready when developers begin to implement it. Prototyping and testing the feature often dictates whether the feature is fit for the game. If the feature is deemed to fit, implementing the feature in to the game will proceed as will the design of the feature. However, developers may not have full understanding of the feature as the design is incomplete, and it may still change. If the designs changes, the implementation may not fully fit the design when it is ready, introducing technical debt:

> All changes in design are reflected everywhere else, so those design changes are the ones that most often then cause technical debt, and at worst they can be reflected in a lot of places. One small change is quite a critical thing. (**I3**)

I4 described that as the design progresses, it may also expand the feature, making it more significant factor in the game. As the developers do not have the full picture of the feature at the start of the implementation, it may not be modular enough for the future purposes.

Another common factor for technical debt accumulation stated by the interviewees was inadequate testing. Most of the interviewees stated that testing in game development is ad-hoc, lacking any systematic way to test developed features. Developers often test their own features by trying the feature in the game instead of systematic unit testing. I2 stated that developers may not test the feature as a part of larger concept. Most of the interviewees also stated that test automation is hard in game development, thus playing a much smaller role. Inadequate testing introduces testing debt.

Lastly, I5 pointed out that avoiding technical debt is not an important factor if the future of the game is uncertain:

> In principle, when a game is being developed, it gets released and then it's live - hopefully, let's say, for a decade. The situation of the game is such that when you're trying to get the game out, if things go badly, then the game is kind of fighting for its existence. At that point you're not thinking about technical debt at all, you're just trying to get things to happen - so in a way, technical debt doesn't matter if the game is dead tomorrow. (**I5**)

4.6 Results on Technical Debt Management

The last section of results, technical debt management, comprises of three different questions; Q7, Q8, and Q9. Interviewees' answers give insight how technical debt is managed, how it is prevented and whether it is documented in game development.

When interviewees were asked about how technical debt is managed in their job (Q7), they did not report any systematic approaches to technical debt management. Interviewees I3 and I4 reported that usually debt management starts with a developer noticing an issue or limitation in the system under development, who then reports it onward:

> In my work it's often like that you hit some kind of limit somewhere - it might be that an individual developer says that okay, if we now want to do the things we've planned, then at this point we basically have to refactor some system, because otherwise we'll end up with too much technical debt, that at some point we just have to clean that debt out. (**I4**)

An issue could show as repeated slowness or difficulty in changing a part of a system. According to I1, it is then evaluated whether it is worth it to refactor the old implementation or create completely new one. Multiple interviewees reported that

usually only the issues which cause continuous problems in development or significant performance problems, are considered for repayment. Interviewee I3 explained that also a change in design should also prompt a refactor of the implementation, if that change is significant enough. I4 said that it depends on the developers how much they would like to use time to repay technical debt back instead of developing new features. According to them, the direct benefits to players from developing new features outweigh the indirect benefits from refactoring debt-ridden implementations. I5 also explained that there is never a good time for refactoring: when the game is not doing so well, it is "kind of fighting for its existence", but even when the game is thriving there is no time for refactoring:

> Because this is such a hit-driven business and everyone knows that if things are going well today, then next week they probably won't be going as well, so now you kind of have to... when the iron is hot, you have to strike. And then at that point you want to produce as much content for players as possible, new events, whatever you can make, so that you can keep that peak going as long as possible. So then again, even then it's not really the right time - it's not seen as a good moment to start looking at technical debt and such. So I'd see it as kind of like there's always, always a rush to the next thing, and so the end result is always kind of the number one priority. (**I5**)

I5 suggested that instead of reactive technical debt management, developers should be given more time at the start of the project to implement features the right way first time around, so development teams could avoid delays caused by technical debt.

With the next question, the Question Q8, interviewees were asked to tell about what methods they use to prevent debt accumulation in their daily work. Interviewees answered with some software engineering practices that could help avoid

technical debt. I1 explained that adjusting scope and prioritizing major features is the most effective way to prevent debt accumulation. Developers should prioritized creating features that are most essential to the core gameplay in the game. Major features should be given enough time to do them the right way, avoiding larger refactors to them. They also stated that developers should cut features that are deemed unnecessary, and simplifying those features that are wanted in the game, but do not play a major role. Adjusting scope is used to avoid degradation of software quality, sacrificing scope size rather than software quality.

Interviewee I3 highlighted that developers should try to avoid over-engineering solutions at early stages of development. The developers should not created over-complicated features by trying to see into the future when design is not mature or complete enough. Over-complicated features are harder to refactor if they do not fit the design in the later stages of development. Instead, developers should create smaller, more modular systems that are easier to refactor, lessening the impact of technical debt accumulated in the implementation. Interviewee I4 also endorsed more flexible solutions that are minimally interwoven with other parts of the system. I4 also encouraged to aim for more reusable code. I3 continued that creating placeholder solutions that are easier to replace should be preferred when the design and requirements are still evolving. Lastly, they said that adhering to programming paradigms helps developers in keeping technical debt to the minimum.

Again, I5 highlighted the importance of giving developers enough time to create better solutions. They stated that developers can make better decisions with more time and more mature designs, avoiding the drawbacks that technical debt may cause in the future.

Every interviewee agreed at the last question, Question Q9, that they do not use any official or systematic way to document instances of technical debt in the project. I4 explained that debt instances which cause the most issues are often

informally talked about between the developers. Most of the interviewees said that technical debt instances may be added as tickets in issue-tracking software by the developers who have encountered the debt. I1 added that the ticket should include description of how the system is currently working and how it should work. These tickets may then be presented in sprint planning, and hopefully, a developer has time to address it. Otherwise the ticket stays in the system. Interviewees explained that they usually do not have the time or the interest in documenting debt well, as more pressing matters are prioritized.

5 Discussion

The goal of this chapter is to discuss the results of interviews and their relationship with information learned in Chapters 2 and 3. Discussion in this chapter is structured based on the research questions RQ1, RQ2, and RQ3 established in Chapter 1.

5.1 Technical Debt Accumulation

In this section the first research question, RQ1, is discussed. The interview results on accumulation of technical debt is compared to the characteristics found in Section 3.2. The author of the thesis discusses which characteristics are similar to interview results, what differences are observed, and lastly what new reasons for technical debt accumulation are identified.

The characteristics that align with the results of interviews are as follows: requirements, change, and planning; sub-par testing methods; and lack of documentation. The requirements of game development was not mentioned once in the interviews. Interviewees did not discuss issues about what requirements should be set for a game as a whole. Even if requirements were not mentioned, it could be deduced that the requirements for the implementation of game features come from the design of the game. As the design changes, the implementation requirements change at the same time. Changing design can then create situations where already implementations do not fit the design, creating technical debt to the implementa-

tion. The nature of prototypes described by interviewees is not exactly the same as described in the Section 3.2.1. Interviewees reported that the prototypes should be discarded when the prototype feature is accepted as part of the game, but prototypes are described to be the basis for further development by default in the literature and any technical debt in them is either paid back or left in it. The key difference is that interviewees said that prototype code may be included in the project if the development schedule does not allow developers to discard the prototype and start over with the implementation. If the prototype code is used in the feature, any technical debt will then be moving on to the project. Changing design and changing requirements for the implementation was a common reason stated for technical debt accumulation.

Similar characteristics to sub-par testing methods found in Section 3.2.2 was described by the interviewees. Test debt and test automation debt is accumulated because of disorganized testing in game development. Interviewees said that most of the testing is ad-hoc in nature, developers testing their own implementations. Test automation was stated to often be too difficult to properly test the game. Interviewees did not state any testing strategies, nor any test planning. Inadequate testing was occasionally mentioned by the interviewees.

Most of the problems with documentation stated in Section 3.2.3 was also present in the answers of interviewees. Documentation is often outdated, and it is not seen as beneficial to the progress of development, especially when the documentation would have limited use. At least one interviewee mentioned problems caused by lack of knowledge sharing and documentation, which is called knowledge vaporization in Section 3.2.3. Inadequate methodologies in game development was not discussed by interviewees. Lack of documentation was often mentioned by the interviewees, but technical debt caused by lack of documentation was not common.

Last three characteristics out of the total six in Section 3.2 that was either

not mentioned by the interviewees at all or did not fit well were the following: monetization over software quality, dormant debt, and crunch time. Interviewees did not say that developers would be more prone to accumulate technical debt, if the profitability goals of the game are met. Interviewee I5 mentioned that developers may accumulate more technical debt if the future of the game in development is uncertain. This would indicate that at least the opposite is true: If the game project is not doing well financially, developers could accumulate more technical debt to give the game a fighting chance to survive.

None of the interviewees mentioned to have been a part of a game development project, where dormant debt would have been a factor, at least not in the way described in Section 3.2.5. Interviewee I4 mentioned to have been working in a project that was in development for a long time, that has a lot of technical debt accumulated in it. However, it does not fit the description of dormant debt that is revived when project that is not actively developed is revived.

Last characteristic, crunch time, was not brought up by interviewees. Interviewees stated that limited resources and lack of time was one source of technical debt accumulation, but they did not state that crunch time, working periods of overtime, was a factor to technical debt accumulation.

Lastly, interviewees stated reasons for technical debt accumulation that were not discussed previously in the thesis. Most common reason for technical debt accumulation was limited resources, either in time, personnel, or budget. Interviewees commonly stated that technical debt was caused by lack of time to implement features well or create documentation, for example. Some artifacts, for example most of the quality assurance and control artifacts, could accumulate technical debt because project's budget would not allow to hire dedicated personnel or outsource for the creation of the artifacts. Poor implementation were occasionally stated as the source of additional debt. Developers would have to further develop features on top of tech-

nical debt ridden features, as they would not have the time to repay the previous technical debt, growing the amount of debt in the project. Last reason for technical debt accumulation was developers assuming tasks and duties of multiple different roles, especially in smaller teams. Interviewees stated that developers would either lack the dedicated personnel for certain roles, or the dedicated personnel would not have the time to do all of their tasks, making unqualified personnel assume tasks from roles outside of their qualifications or usual tasks. These situations would create more technical debt due to the limited knowledge about best practices needed for the tasks. Developers assuming multiple roles was common in smaller teams, but uncommon in larger ones.

5.2 Technical Debt in Artifacts

In this section, the second research question, RQ2, is discussed. The interview results on technical debt in game development artifacts are compared to the Table 3.8 to discuss which artifacts were expected to have technical debt and what difference there are between the results and expectations. However, due to the limitations in the interviews, it could not be verified whether the characteristics were the causes of technical debt, even when interviewees indicated that an artifact could contain debt.

In the design artifacts, only the Artifact A1.4 Game Design Document (GDD) was strongly indicated to have technical debt in it, with four out of five interviewees stating that the artifact could have technical debt in it. The artifact was identified to be associated with technical debt in requirements, change, and planning; and lack of documentation and methodologies characteristics. Artifact A1.2 Concept Document was stated to have debt by one interviewee, and in A1.3 Game Bible by two interviewee, with one of them being unsure. A1.2 was not identified to be associated to any of the characteristics, but A1.3 was associated with lack of documentation

characteristic. Interviewees did not see the Artifact A1.1 Brainstorming Document to have technical debt, and interviewees did not have experience with A1.5 Research Materials.

In engineering artifacts, A2.4 Source Code and A2.7 Technical Design Document were both reported to have technical debt in them by most of the interviewees. Characteristics that could be associated with Artifact A2.4 were requirements, change, and planning; and dormant debt. For Artifact A2.7, the characteristics were requirements, change, and planning; and lack of documentation and methodologies. The Artifact A2.1 API Documentation had different results based on the background of an interviewee: interviewees working in smaller studios indicated that A2.1 did not have debt in it, but interviewees working in large studio indicate that it could have debt. In the characteristic lack of documentation and methodologies, Artifact A2.1 was identified to potentially have technical debt. Most interesting artifact in the engineering section was A2.6 Prototypes which was found to potentially have debt in strongly associated to characteristic requirements, change, and planning. Despite the strong association to debt in the characteristic, interviewees did not see the artifact to have technical debt. The reason why interviewees did not agree with A2.6 having debt is that when it is correctly used, it should be discard and therefore it should not have technical debt. Three of the interviewees stated that the artifact could cause debt, if it is incorrectly used, and the artifact is not discarded. At last, the Artifact A2.3 Build notes was stated by four out of five interviewee that it may have technical debt. As it is documentation artifact and tied to the lack of documentation characteristic, it would be likely that the artifact could have technical debt. The rest of the artifacts in the engineering section were inconclusive whether they could have technical debt.

In the next section, quality assurance and control artifacts, A3.3 Performance Tool was found to be the only artifacts that would most likely have technical debt in

it. Artifact A3.6 Test Automation Artifacts could also have technical debt in it, but only one interviewee has experience with the artifact. The Artifact A3.3 was not linked to any of the characteristics, but A3.6 was linked to sub-par testing methods.

Interviewees experience only with the Artifact A4.2 Art Asset to state that the artifact could potentially have debt in the art artifacts. All of the other artifacts in the section were unfamiliar to the interviewees, so they could not comment on them with confidence. A4.2 was not associated with any of the characteristics.

Artifacts in the next section, audio artifacts, were not familiar to the interviewees at all to make statements about the potential technical debt in them.

In writing artifacts, Interviewee I2 statements could hold more weight than the other interviewees, as they are the only interviewee with narrative designer as their role. The interviewee stated that almost all of the artifacts could contain debt as they are mostly documentation artifacts about the games narrative and writing. Artifacts A6.2 Localization Document, A6.3 Setting document, A6.5 Story Bible, and A6.6 Narrative Design Document were related to the lack of documentation characteristic.

At the last section of artifacts, production artifacts, the results were mixed. Artifact A7.1 Asset Archive was reported by two interviewees to have technical debt in it, with one interviewee being unsure. A7.7 Schedule was also stated by two interviewees to have debt in it, one interviewee stated that it could cause debt, and two interviewees did not asssociate it with debt. A7.6 Postmortem was concluded to not have debt in it, and the rest of the artifacts were inconclusive whether they would have debt in them. None of the artifacts were associated to any of the characteristics mentioned.

Most common technical debt type in the artifacts is documentation debt. Game development projects include multiple documentation artifacts from multiple different areas. Many of the living documentation artifacts were prone to have technical

debt at some point of the development. Documentation artifacts that are not meant to be further updated after creation, did not fit the definition of technical debt. Design, code, and architecture debts were present in the interviewees' answers about technical implementations. Many of the interviewees reported facing issues with implementation that do not fit the design, code that is hard to understand and change, and monolithic and complex structures in the implementations. Test debt was present as lack of test planning, ad-hoc testing by the developers, and usage of test automation being uncommon, introducing test automation debt into the project. Interviewees also mentioned that some tasks in the issue tracking software could stay in for years, which would count as defect debt.

5.3 Management of Technical Debt

In the last section of this chapter, the third research questions, RQ3, is discussed. Results from the interviews on technical debt management are examined for which of the eight TDM activities are present or missing, how debt is prevented, how it is documented, and whether any proposed TDM strategies are present in game development.

Interviewees reported that technical debt management usually starts with a developer noticing a reoccurring issue which complicates development. Developer noticing an issue could count as identification of debt. After identifying technical debt in the system, it is then documented and communicated to other developers, usually in issue tracking system and/or in sprint planning. Identified and communicated debt instances are evaluated whether the debt is worth repaying. Repayment of debt could either be refactoring the system already in place or creating completely new system to replace it or to be used along the side of old system. Debt repayment can also be initiated by extensive change in design. Last activity identified from the results of the interview may not be a conscious activity that developers

practice: prevention of technical debt. Instead, they are mostly general software development practices that are used manage software projects and maintain software quality, with the added benefit of reduced amount of technical debt. These general activities are adjusting scope, prioritizing major features, avoiding over-engineering solutions at early stages of development, adhering programming paradigms, and creating smaller, modular solutions that are easier to refactor.

Interview results lack completely three TDM activities: measurement, prioritization, and monitoring. Interviewees did not mention whether it technical debt instances are measured for benefits and drawbacks, but they are mostly identified and considered for repayment when the drawbacks outweigh the benefits. Technical debt instances are not prioritized among themselves, but task created to issue tracking system may be prioritized, which would not count as TDM activity. Lastly, interviewees did not state that debt instances would be monitored for a change.

Interviewees stated that documentation of technical debt instances are poor. Debt instances are added to issue tracking system at most, and are sometimes informally discussed between developers. It is clear that technical debt management strategies presented in Section 2.3.2 are not utilized. Based on the results, technical debt management in game development is non-systematic, need-based reactive management, that is mostly practiced when development speed and difficulty is affected. The interest of technical debt may cost the developers significant amount of time in the current approach to technical debt management. Developers do not manage technical debt until the effects of the debt is significant enough, and smaller instances of technical debt could go without being managed for longer periods of time.

6 Conclusions

This last chapter of thesis focuses on summarizing the answers to research questions discussed in the Chapter 5, the limitations of the research, and what could future research focus on or improve.

6.1 Summarizing Research Questions

RQ1: What are main reasons behind technical debt accumulation in game development?

Most common reason was found to be limited resources. Limited resources meant that development teams did not have sufficient amount of time to properly implement features, create designs, and documentation. Limited budget, especially in smaller teams, often leads to lack of expertise in all of the development areas due to lack of dedicated personnel for each role. Incomplete or changing design could accumulate debt in feature implementations that are started before complete knowledge of the feature is known. This could lead to mismatch between requirements for the feature implementation and the design of the feature. Missing or incomplete documentation is common in game development, as developers would rather create new features for the game rather than documentation. Inadequate documentation and knowledge sharing could lead to situations where complex systems are not fully understood, introducing technical debt in solutions that are based on the complex systems.

RQ2: Which artifacts contain technical debt in game development projects?

Almost every section of development artifacts were found to contain artifacts that could contain technical debt. Most common artifacts were found to be documentation artifacts, but all of the technical debt types presented in Table 2.1 of Section 2.2 were present in the results. Technical debt was also discovered to be in artifacts that are not common in traditional software development, for example in writing artifacts. Some difference between development team sizes could be detected, but due to smaller number of interviewees, anything certain could not be established.

RQ3: How is technical debt managed in game development?

Technical debt management is often disorganized in game development. TDM in game development currently lacks systematic approaches to manage, document and prevent technical debt. Technical debt management is typically undertaken when the speed and ease of development has already declined, making the management most of the time reactive rather than proactive. Any technical debt instances identified during development are poorly documented, unless it is affecting development significantly.

6.2 Limitations of the Research

Some limitations were discovered during the writing of this thesis. First, and the most signification limitation discovered is the lack of diversity in roles of the interviewees. Four out of the five interviewees' field of expertise were in focused in software development, which negatively affected the interview results on the artifacts. The small sample size and limited representation of different roles result in unclear conclusions about which artifacts may exhibit technical debt, as the interviewees lacked sufficient experience across areas of development artifacts outside of their expertise. However, the results did nonetheless provide insight to which artifacts could have

debt, what type of debt is common, and that every section of development artifacts could have technical debt.

Second limitation was the thesis' author's inexperience as an interviewer. During processing of interview data, the author noticed some instances where the answer of the interviewee was not fully clear, and the author could have asked for elaboration during the interviews. Generally interviewees' answers were sufficient, but more insight could have been gained with more experience in interviewing.

6.3 Future Research

Like mentioned in the Section 1.1, the game development industry could benefit from more research. The outcome of this thesis revealed that further research could be conducted in technical debt management in game development. As the current state of TDM in game development is disorganized, future research could focus on TDM activities, and strategies. Researcher could study which technical debt management strategies could fit game development, to lessen the time wasted on the effects of technical debt. Like mentioned in the previous Section 6.2, future research could benefit from a larger number of interviewees, with more diverse disciplines to better explore artifacts in all of the sections of development artifacts.

References

[1] 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*, ser. ICSE 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 1–11, ISBN: 9781450327565. DOI: 10.1145/2568225.2568226. [Online]. Available: https://doi.org/10.1145/2568225.2568226.

[2] N. Wirth, "A brief history of software engineering", *IEEE Annals of the History of Computing*, vol. 30, no. 3, pp. 32–39, 2008. DOI: 10.1109/MAHC.2008.33.

[3] P. Hohl et al., "Back to the future: Origins and directions of the "agile manifesto" – views of the originators", eng, *Journal of Software Engineering Research and Development*, vol. 6, no. 1, pp. 1–27, 2018, ISSN: 2195-1721.

[4] T. Teubner, C. M. Flath, C. Weinhardt, W. van der Aalst, and O. Hinz, "Welcome to the era of chatgpt et al: The prospects of large language models", eng, *Business & information systems engineering*, vol. 65, no. 2, pp. 95–101, 2023, ISSN: 2363-7005.

[5] T. Wijman, *Last looks: The global games market in 2023*, [Accessed 08-12-2024], May 2024. [Online]. Available: https://newzoo.com/resources/blog/last-looks-the-global-games-market-in-2023.

[6] "Industry data". [Accessed 08-12-2024]. [Online]. Available: https://www.ifpi.org/our-industry/industry-data/.

[7] "Cinema - Worldwide | Statista Market Forecast — statista.com". [Accessed 08-12-2024]. [Online]. Available: https://www.statista.com/outlook/amo/media/cinema/worldwide?currency=USD.

[8] W. Cunningham, "The wycash portfolio management system", *SIGPLAN OOPS Mess.*, vol. 4, no. 2, pp. 29–30, Dec. 1992, ISSN: 1055-6400. DOI: 10.1145/157710.157715. [Online]. Available: https://doi.org/10.1145/157710.157715.

[9] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt", eng, *The Journal of systems and software*, vol. 86, no. 6, pp. 1498–1516, 2013, ISSN: 0164-1212.

[10] N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners", *Information and Software Technology*, vol. 102, pp. 117–145, 2018, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.05.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918300946.

[11] N. Rios, L. Ribeiro, V. Caires, T. Mendes, and R. Spínola, "Towards an ontology of terms on technical debt", *Proceedings - 2014 6th IEEE International Workshop on Managing Technical Debt, MTD 2014*, pp. 1–7, Dec. 2014. DOI: 10.1109/MTD.2014.9.

[12] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management", eng, *The Journal of systems and software*, vol. 101, pp. 193–220, 2015, ISSN: 0164-1212.

[13] N. S. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic

mapping study", eng, *Information and software technology*, vol. 70, pp. 100–121, 2016, ISSN: 0950-5849.

[14] A. Martini, V. Stray, and N. B. Moe, "Technical-, social- and process debt in large-scale agile: An exploratory case-study", in *Agile Processes in Software Engineering and Extreme Programming – Workshops*, R. Hoda, Ed., Cham: Springer International Publishing, 2019, pp. 112–119, ISBN: 978-3-030-30126-2.

[15] C. Seaman et al., "Using technical debt data in decision making: Potential decision approaches", eng, in *2012 3rd International Workshop on Managing Technical Debt, MTD 2012 - Proceedings*, IEEE, 2012, pp. 45–48, ISBN: 9781467317481.

[16] Y. Guo and C. Seaman, "A portfolio approach to technical debt management", eng, in *MTD'11 - Proceedings of the 2nd Workshop on Managing Technical Debt, Co-located with ICSE 2011*, New York, NY, USA: ACM, 2011, pp. 31–34, ISBN: 1450305865.

[17] B. J. Wardyga, *The video games textbook : history, business, technology*, eng. Boca Raton: CRC Press, 2023, ISBN: 1-00-331575-5.

[18] J. Chueca, J. Verón, J. Font, F. Pérez, and C. Cetina, "The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games", *Information and Software Technology*, vol. 165, p. 107 330, 2024, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2023.107330. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584923001854.

[19] C. McDonald et al., "Describing, organizing, and maintaining video game development artifacts", eng, *Journal of the American Society for Information Science and Technology*, vol. 72, no. 5, pp. 540–553, 2021, ISSN: 2330-1635.

[20] J. H. Lee et al. "Taxonomy of video game development artifacts". version 1.1. [Online]. Available: https://github.com/uwgamergroup/taxonomy-game-development-artifacts. (accessed: 12.11.2025).

[21] E. Zabardast, J. Gonzalez-Huerta, T. Gorschek, D. Šmite, E. Alégroth, and F. Fagerholm, "A taxonomy of assets for the development of software-intensive products and services", *Journal of Systems and Software*, vol. 202, p. 111 701, 2023, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2023.111701. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223000961.

[22] K. Borowa, A. Zalewski, and A. Saczko, "Living with technical debt—a perspective from the video game industry", *IEEE Software*, vol. 38, no. 6, pp. 65–70, 2021. DOI: 10.1109/MS.2021.3103249.

[23] R. Alfayez, R. Winn, Y. Ding, G. Alfayez, and B. Boehm, "Technical debt (td) through the lens of twitter: A survey", *Journal of Software: Evolution and Process*, vol. 36, no. 4, e2547, 2024. DOI: https://doi.org/10.1002/smr.2547. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2547. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2547.

[24] M. Lehtonen, C. Lu, T. Nummenmaa, and J. Peltonen, "Adoption of requirements engineering methods in game development: A literature and postmortem analysis", in *Interactivity, Game Creation, Design, Learning, and Innovation*, A. Brooks and E. I. Brooks, Eds., Cham: Springer International Publishing, 2020, pp. 436–457, ISBN: 978-3-030-53294-9.

[25] J. Kasurinen, A. Maglyas, and K. Smolander, "Is requirements engineering useless in game development?", in *Requirements Engineering: Foundation for*

Software Quality, C. Salinesi and I. van de Weerd, Eds., Cham: Springer International Publishing, 2014, pp. 1–16, ISBN: 978-3-319-05843-6.

[26] C. Politowski, F. Petrillo, and Y.-G. Guéhéneuc, "A survey of video game testing", in *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, 2021, pp. 90–99. DOI: 10.1109/AST52587.2021.00018.

[27] F. Petrillo, M. Pimenta, F. Trindade, and C. Dietrich, "What went wrong? a survey of problems in game development", *Comput. Entertain.*, vol. 7, no. 1, Feb. 2009. DOI: 10.1145/1486508.1486521. [Online]. Available: https://doi.org/10.1145/1486508.1486521.

[28] H. Edholm, M. Lidström, J.-P. Steghöfer, and H. Burden, "Crunch time: The reasons and effects of unpaid overtime in the games industry", in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2017, pp. 43–52. DOI: 10.1109/ICSE-SEIP.2017.18.

[29] J. I. Trasobares, Á. Domingo, R. Casamayor, D. Blasco, and C. Cetina, "A multiple case study on reuse in game software engineering", eng, *Information and software technology*, vol. 185, pp. 107 781–, 2025, ISSN: 0950-5849.

Appendix A Interview Questions

Pre-interview demographic questions

Q1: What is your job title?

Q2: How long have you been working in software development overall? *(in years)*

Q3: How long have you been working in game development? *(in years)*

Overview of Technical Debt

Q4: How often do you encounter technical debt in your job?

Debt in Game Development Artifacts

A taxonomy of artifacts established in Section 3.1 were presented to interviewee section by section

Q5: In which of these artifacts have you encountered technical debt?

Accumulation of Technical Debt

Q6: What factors affect accumulation of technical debt?

Technical Debt Management

Q7: How technical debt is managed in your job?

Q8: What methods are used to prevent technical debt accumulation?

Q9: How technical debt is documented if technical debt is detected?