NTNU – Trondheim
Norwegian University of
Science and Technology

Software Architecture and the Creative Process in Game Development

Njål Nordmark

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Alf Inge Wang, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The goal of this master thesis is to investigate the relationship between software architecture, the creative team, and the development processes, providing insight into how these relationships are today.

Primarily the thesis will look at how the different game companies use software architecture to facilitate the creative team's work. The thesis will also investigate the effect the different development processes have on the creative team. In addition, the thesis will look at to what extent the team is given the opportunity to contribute in these processes.

Assignment given: January 16th, 2012
Supervisor: Professor Alf Inge Wang, IDI

Abstract

The goal of this thesis has been to perform research on the relationship between the creative team, software architecture, and game development processes.

Researching this relationship was done in three stages. The first stage was a literature review into software architecture and game development. In the second stage a questionnaire was designed based on the literature review, and this questionnaire was then distributed to several game developers. In addition to querying the game developers on their knowledge on the field, they were also asked whether or not they would be willing to answer a set of follow-up questions later.

The responses to the questionnaire provided a lot of answers, but also gave rise to new questions. In the third stage these new questions were incorporated into a follow-up survey which was distributed to those respondents whom had previously answered that they were willing to answer follow-up questions.

The problem definition has been divided into five research questions according to the Goal Question Metric approach. Supported by the literature review and the responses to both the questionnaire and the survey, these five research questions have been answered in detail in Chapter 11: "Research Conclusions".

The results from this thesis is not generalizable to all game developers, but provides a very interesting glimpse into how the creative team is affected by, and is allowed to affect, the software architecture and tools used, as well as the game development process.

Samandrag

Målet med denne masteroppgåva har vore å forska på forholda mellom dei som jobbar med dei kreative aspekta av spel, programvarearkitekturen som vert brukt og prosessane rundt spelutvikling.

Denne forskinga vart gjennomført i tre stadium. Det fyrste stadiet var eit litteraturstudium som fokuserte på programvarearkitektur og spelutvikling. I det andre stadiet av forskinga vart det utforma ei spørjeundersøking som var basert på litteraturstudiet. Denne spørjeundersøkinga vart så send ut til mange forskjellege spelutviklarar. I tillegg til å stilla spørsmål til spelutviklarane om spelutvikling og dei kreative prosessane, vart dei og spurde om dei kunne svara på nokre oppfølgingsspørsmål.

Svara på spørjeundersøkinga var til stor hjelp når konklusjonen på forskingsspørmåla skulle utformast, men gav og grunnlag for ein del nye spørsmål. I det tredje stadiet av forskinga vart desse nye spørsmåla tatt med i ei oppfølgingsundersøking. Denne undersøkinga vart så sendt ut til dei som gjennom spørjeundersøkinga hadde sagt at dei var viljuge til å svara på oppfølgingsspørsmål.

Problemdefinisjonen har vorte delt opp i fem separate forskingsspørsmål ved bruk av Goal Question Metric-tilnærminga. Understøtta av litteratur -studiet og svara på både spørjeundersøkinga og oppfølgingsundersøkinga vert desse forskingsspørsmåla gjord greie for i detalj i Kapittel 11: “Research Conclusions”.

Resultata frå denne forskinga kan ikkje generaliserast på ein slik måte at dei vert gjeldande for alle spelutviklarar. Trass i dette gjev resultata eit godt innblikk i korleis dei kreative aspekta ved spelutvikling både påverkar og vert påverka av programvarearkitekturen og verktya som vert brukte, samt spelutviklingsprosessen.

Preface

This report is the master thesis delivered in the course *TDT4900 - Computer and Information Science, Master Thesis*, marking the end of a five year master's program at the Department of Computer and Information Science under the Faculty of Information Technology, Mathematics and Electrical Engineering at the Norwegian University of Science and Technology.

The researcher would like to thank professor Alf Inge Wang for his continuous support and enduring enthusiasm during the process of working with this thesis, providing the researcher with the opportunity to study the field of software architecture and games.

The researcher would also like to thank all of those who have given their comments on the thesis, in addition to those who have proofread the questionnaire, the survey and the thesis itself.

Trondheim, June 5th, 2012

Njål Nordmark

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

FPS First-Person Shooter

GQM Goal Question Metric

IDI Department of Computer and Information Science

IME Faculty of Information Technology, Mathematics and Electrical Engineering

MMOG Massive Multiplayer Online Game

NFR Non-Functional Requirement

NPC Non-Player Character

NTNU Norwegian University of Science and Technology

OS Operating System

RQ Research Question

XML Extensible Markup Language

Part I

Introduction

Chapter 1

Project Introduction

In this chapter a short introduction to the motivation behind the project and the project itself is given.

1.1 Project Context

This project is articulated by professor Alf Inge Wang at Norwegian University of Science and Technology (NTNU). It is a master thesis in the game research program at Department of Computer and Information Science (IDI) under the Faculty of Information Technology, Mathematics and Electrical Engineering (IME).

It is a research spurred from a project performed in the fall of 2011 (Nordmark [18]), and will look further into the roles of a developing organization and how these affect the software architecture of a game. The problem definition is presented in Section 1.3.

1.2 Project Motivation

The video game industry is today a major part of the software development industry as well as the entertainment industry. It has been an area of intense growth since the beginning in the 1970s [29], and is in many ways just beginning to mature into the engineering-aspect of software engineering.

In other areas of software engineering, the developers and companies are starting to look towards how to enable simple maintenance, reuse, and further development through best-practices both in terms of processes as well as the technical aspects, such as software architecture. As presented in the book by Rollings and Morris [21], the authors are also worried that the video game industry is lagging behind the rest of the software industry.

In the research Nordmark [18] it is discovered that the game developers have a concious relationship to software architecture, but that they are

lacking in other areas. These findings suggested new areas of interest, particularly how the creative team[1] affects, and are affected by, the software architecture in the underlying game or game engine.

1.3 Problem Definition

The main goal of this research entitled "Software Architecture and the Creative Process in Game Development" is to perform research into software architecture in the field of game development. The study will look at three different aspects of this; (1) how are game developers today using software architecture to facilitate the creative processes in game development, (2) which methods and practices do they use to make the creative team's job easier, and (3) are there ideas from other areas of software engineering which can be used to promote the creative work in game development.

1.3.1 Creative Team versus Technical Team

In the text above, a distinction between *the creative team* and *the technical team* is introduced. However, the researcher recognizes that most people involved in game development perform creative work in their day-to-day tasks. By using this distinction, it is possible to make the language of the text both more precise and easier to read. By *the creative team* the researcher includes those who work with tasks concerned with game design, artistry, generating content, level design, story, etc. Similarly, *the technical team* includes those who work with implementing the game, game engine, and supporting tools and systems in source code.

Note that these two groups are not necessarily mutually exclusive. Members of the technical team can very well be adept artist, and vice versa.

1.4 Research Questions

Based on the problem definition given in Section 1.3, the following research questions have been identified:

RQ1: What are the primary ways in which the creative team can affect the software architecture in a game?

RQ2: Are there any particular architectural approaches that facilitate the creative processes in game development?

RQ3: Are there any particular development methods or processes which help the creative team do their job?

[1] See Section 1.3.1 for a discussion of the distinction between *the creative team* and *the technical team*.

RQ4: Do the organizations prioritize the needs and wants of the creative team?

RQ5: How has game development evolved over the years as an engineering discipline?

1.5 Structure

This thesis is divided into four main parts. Part I introduces the background for the project, the research method used, and the overall goal of the project. Part II investigates different areas of software architecture and research, laying the foundation for the research. In Part III the actual research of this thesis is presented. The results for on the different questions (metrics in Goal Question Metric (GQM)) is analyzed to enable drawing conclusions. In addition to this a short evaluation of the research as well as future areas of interest is presented. In Part IV conclusions regarding the research questions (see Section 1.4) are presented. The thesis is then concluded with some suggestions for future studies.

1.6 Related Work

Apocalypse Engine

In 2009, Guldbrandsen and Storstein performed a research into software architecture and game engines [13]. This research was done as a preparation for their master thesis at NTNU.

Software Architecture in Games

In 2011, Nordmark performed a research into software architecture in games and how the different roles of the employees in a developing organization can affect the software architecture in the final game [18]. This research was done immediately preceding this master thesis.

Requirements Engineering and the Creative Processes in the Video Game Industry

Callele, Neufeld, and Schneider perform a review of practices of requirements engineering in games, and how these practices affect the development cycle [6]. In particular, Callele et al. address issues related to Non-Functional Requirements (NFRs), and how these requirements should be presented in a requirement document for the technical team.

Postmortem Analysis of Video Games

Kvasbø presents a thorough review of most of the postmortems in *Game Developer Magazine* [24] from 1994 through 2006, with particular focus on the evolution of game development [16]. This research was done as a preparation for his master thesis at NTNU.

Chapter 2

Research Method

In this chapter several key points of performing research will be introduced, including how goals are defined, different strategies for researching those goals, and how to validate the conclusion.

2.1 The Scientific Method for Software Engineering

To perform any kind of research, one needs to follow a scientific method. In the article "The Experimental Paradigm in Software Engineering" [1], Basili discusses two main approaches to researching software architecture; (1) the scientific method and (2) the mathematical method.

Basili defines them as follows:

The Scientific Method: Observe the world, propose a model or a theory of behavior, measure and analyze, validate hypotheses of the model or theory, and if possible repeat the procedure.

The Mathematical Method: Propose a formal theory or set of axioms, develop a theory, derive results and if possible compare with empirical observations.

The mathematical method is better suited for a formalization of an existing procedure which does not necessarily require too much human thought. However, it is difficult to use it to research the creative sides of software engineering. During software development, there is a great deal of creativeness from each software developer, and thus, the scientific method is better aligned with research into software architecture and the processes surrounding it.

Basili subdivides the scientific method into two more specific methods; (1) the engineering method and (2) the empirical method.

The Engineering Method: observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible.

The engineering method is an evolutionary research method. This is important as it usually cannot result in radical changes, but only improve the different objects under study.

The researcher looks at an object of interest (e.g., a process or a tool), and tries to find ways to improve this. This improvement is then implemented and tried out, e.g., using a case study (see Section 2.4.2). Identifying new objects of interest can be done in many ways. One natural way is that the researcher knows the system because she has used it. Another way can be by performing a explorative pre-study, e.g., by using surveys (see Section 2.4.1).

The Empirical Method: propose a model, develop statistical/qualitative methods, apply to case studies, measure and analyze, validate the model and repeat the procedure.

The empirical method is a more revolutionary approach than the engineering method. A researcher proposes a (possibly) new set of ideas (e.g., a set of tools or processes). The researcher then develops measures for this new model, and applies them to a suitable situation, e.g., by using a case study (see Section 2.4.2). If the results are promising, this can be done several times to further improve the new model.

2.2 Goal Question Metric

Regardless of the chosen research method, it is important to have a clearly defined goal of the research, as well as an understanding of how one should fulfill this goal. Basili, Caldiera, and Rombach [2] argue that any measurement of any part of software engineering needs to be defined in a top down fashion. Following this, they identify that for a measurement to be effective, it needs to be:

1. Focused on specific goals
2. Applied to all life-cycle products, processes, and resources
3. Interpreted based on characterization and understanding of the organizational context, environment, and goal

Basili et al. then go on to propose an approach to help create these measurements, an approach called Goal Question Metric (GQM). The GQM approach is divided into three levels, defined as follows:

Conceptual level (GOAL): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment.

Operational level (QUESTION): A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try to characterize the object of measurement (product, process, resource) with respect to a selected quality issue and to determine its quality from the selected viewpoint.

Quantitative level (METRIC): A set of data is associated with every question in order to answer it in a quantitative way. The data can be [objective or subjective][1].

This approach lends itself very well to creating questionnaires. Here one defines the goal, which leads to the research questions, which again leads to questions one can use on a questionnaire.

2.3 Research Paradigms

In the book "Experimentation in Software Engineering: An Introduction" [31] the authors Wohlin et al. present the two main research paradigms in empirical research; qualitative and quantitative research.

Qualitative: A qualitative research is performed on the object of interest in it's natural setting. This is done by gathering descriptions of the object of interest from different sources and then generalizing the descriptions into a conclusion or hypothesis.

Quantitative: A quantitative research is a research in which one tries to identify a relationship between to objects or group of objects. This research is usually performed in a controlled or semi-controlled setting in which the researcher can modify certain values or settings and measure the effect on the object of interest. The relationship can be precise, e.g., a correlation factor, or it can be a more general ordering, e.g., method A is better than method B.

2.4 Empirical Strategies

Wohlin et al. [31] identify three main approaches, or strategies, for collecting data for a research; surveys, case studies, and experiments.

[1] See Section 2.6.1 for a discussion of objective and subjective measures

2.4.1 Survey

A survey is an activity in which one tries to make a snapshot of the current situation. It asks a subset of a population a set of questions regarding their perception and understanding of the object of interest. These answers are then analyzed, generalizing the information relating to the object, enabling assertions or conclusions to be drawn. Surveys are often used in market research and other polls, as they are suitable to perform exploratory research, e.g., by charting new opinions or conceptions.

Wohlin et al. [31] identify three main objectives for conducting a survey:

Descriptive surveys are used to enable assertions regarding a population without considering the reasons for this assertion

Explanatory surveys are used to explain certain phenomenon in a population

Exploratory surveys are often used as a pre-study for another research, allowing the researcher to verify that no key point is missed

2.4.2 Case Study

A case study is an activity in which one tries to measure a single object or phenomenon at a particular time. A strong point of the case study is that it is performed in a semi-controlled setting. In this setting, the researchers are able to control at least parts of the situation, but still allow the participants to act as if everything is normal.

For instance, a case study is a good way to compare a new method or principle with a baseline. If a large company is considering changing it's software development methodology, it may be a good idea to allow one or a few teams to try it out first. If the teams and projects are chosen carefully, making sure that they are comparable to other teams and projects in the organization, the researcher has a good starting point. The team can then use the new methods, and the researcher can record the necessary metrics to compare this new method to the old one.

There are two major limitations to case studies; they cannot be replicated, and they have limited validity. Since a case study is a measure taken at a specific time in a specific situation, there is no way to reconstruct this exact situation at a later time. Because of this, it is difficult to generalize the results of such a case study, since there is no viable, scientific way to validate the conclusions. This again limits the validity of the results. Often the results of a case study can only be used by the group or organization which performed the research, and then again, only for a relatively short period after it was performed.

2.4.3 Experiments

An experiment is a research activity in which the researcher controls every aspect of the situation in which the test subjects are placed. Since the researcher then can vary every single aspect of the situation randomly, it is well suited to confirming theories or conventional knowledge.

As an example, a researcher wishes to test the calculation speed of a certain mathematical operation in different programming languages. She then has control over which developer writes the program, which programming language she uses, which compiler is used, which computer/architecture is used, etc. By varying these variables, the researcher can identify the relationships among them and draw a conclusions.

A great strength of an experiment is that the results are valid far beyond the organization which performed it. Anyone can redo the experiment and check it's validity, and thus it allows for general theories or concepts to be proven.

2.5 Literature Review

As a part of this research, literature review will be used to answers parts of the research questions. A literature review is a process wherein one goes through existing literature on the subject. This is done to give the researcher a thorough understanding of the subject matter, as well as helping to identify gaps in the current knowledge where further research is appropriate [5].

2.6 Measurements in Software Engineering

In Wohlin et al. [31] an overview of scientific measurement focused on measurement of software and software engineering is presented.

First and foremost, Wohlin et al. argue that any scientific measure must be a valid measure. By valid, they mean a measure that does not violate any necessary properties of the object, and that the measure correctly characterize the object mathematically. Furthermore, any statements made regarding this measurement (conclusions or generalizations) must be what they term "meaningful". A meaningful statement is not invalidated by a change of measurement scale (e.g., a change from Celsius to Fahrenheit).

The authors also discuss different types of measurement scales, and identify the following four different scales:

Nominal Scale: A nominal scale is a one-to-one mapping form the measure to the scale. This makes it difficult to conclude regarding internal ordering within the measurement.

Ordinal Scale: A ordinal scale is used when the measurements can be sorted by an ordering criterion. This scale also includes the relative distance between the measurements.

Interval Scale: An interval scale is used when the value distance between two measurements are useful, but not the value in itself.

Ratio Scale: A ratio scale is used when there is a meaningful zero value of the scale, and the ratio between two measurements can have a meaning.

Chiefly, the nominal and ordinal scale are used when doing qualitative research, and the interval and ratio scale are used when doing quantitative research.

Furthermore, the authors discuss two other aspects of a measure; objective versus subjective and direct versus indirect.

2.6.1 Objective and Subjective Measure

A measure is either objective or subjective. An objective measure is a measure which is only dependent upon the object under measurement. A subjective measure is a measure wherein the person measuring the object uses judgement when measuring.

2.6.2 Direct or Indirect Measure

A direct measure is when one can measure the value directly from the object. An indirect measure is calculated using other measures.

2.7 Validation of Results

As an important part of any research, the results will need to be validated. Wohlin et al. [31] state that "adequate validity refers to that the results should be valid for the population of interest".

This implies two things; (1) the researcher needs to consider which population she would like to make a generalization about and (2) the researcher needs to modify the research in a way which allows this generalization to happen.

First and foremost the researcher should decide which population she wants the results to apply to. Based on this, the researcher can choose a suitable subset on which the research is to be performed. When the subset is decided, the researcher needs to consider if this is a representable selection, and if the results from this group can actually be valid for the entire population of interest.

After the researcher has found a subset and concluded on which population the research can be applied to, she can make the research specific for this group. If the research is a survey or questionnaire, the questions can be adapted to the selected population, making the results as applicable as desired.

2.8 Application of Research Methods

In this section, the application of the different research methods to this research, as well as the rationale for choosing them, will be presented.

2.8.1 Research Method and Paradigm

The research is tasked with proposing models for how software architecture can be used to facilitate the creative processes in game development. From this model, a subset of game developers will be studied to validate the model.

This research is a qualitative one, and suites the empirical method described by Basili [1].

2.8.2 Problem Definition

The presented GQM approach will be used to define the problem at all the necessary levels.

Firstly, the overarching research goal will be defined (as the conceptual level in GQM). This goal is presented in Section 1.3. Then, the Research Questions (RQs) will be defined to support a conclusion to the research goal (the operational level). These are presented in Section 1.4. Lastly, the specific questions for the questionnaire and survey will be defined, allowing conclusions to be drawn regarding the RQs (quantitative level). The questionnaire is presented in Chapter 7, and the survey in Chapter 8.

2.8.3 Empirical Strategies and Measurements

During this research, there are two main phases; (1) a pre-study on technical solutions and processes geared at helping the creative team in the game company, and (2) the main study where real-world data is gathered and analyzed, comparing this data to the results of the pre-study.

During the research, a questionnaire will be sent out to game developers. The rationale for using a questionnaire instead of a full-fledge survey, is that it reduces the effort needed for answering. This will provide an interesting glimpse into how different game developers are doing their work today.

After the questionnaire has been completed, a normal survey will be sent out to a subset of game developers. This survey will be adapted to the

responses to the original questionnaire, enabling the researcher to conclude on the five RQs.

Given the large discrepancy between different game developers, combined with the pure qualitative research being performed, the results will map to a nominal scale.

The measures will also be subjective measures, as the replies to both the questionnaire and the survey will be subject to the researcher's interpretation.

Part II

Pre-Study

Chapter 3

Software Architecture

In this chapter an introduction to software architecture will be given. First, an understanding of what software architecture is is given. This is followed by a short discussion of the goals a software architecture can have. Lastly, two different methods for designing software architecture is presented.

As a complete overview is beyond the scope of this thesis, this introduction will be necessarily brief.

3.1 What is Software Architecture?

Before this research can be performed, a thorough understanding of what the term "software architecture" means is needed.

A good starting point for anyone studying software architecture, is the article "Foundations for the Study of Software Architecture" [19] by Perry and Wolf. This article presents some history of the field, as well as an intuitive understanding of software architecture.

The starting point for most major fields of study is a need of solutions to existing problems. In the 1960s developers encountered new problems in the computer science field, where they were to develop large-scale software systems. This had not been done before, and spurred the study of "software design". This in turn became more abstract over the years, up to the point where one can identify it as the study of software architecture.

But what is software architecture?

There are many definitions available today which all propose to be the one definition of software architecture. Perry and Wolf [19] present the following model:

$$\text{Software Architecture} = \{\text{Elements}, \text{Form}, \text{Rational}\}\,.$$

This model is further discussed in their article [19], but there are a few points one should take note of:

Elements: These are the actual pieces that the software architecture consists of. Perry and Wolf divides the elements into (1) processing elements, (2) data elements, and (3) connecting elements.

Form: The form of a software architecture is the constraints imposed on the implementation. This can relate to properties of particular elements (e.g., this element should output its data in XML-format) or the relationships among them (e.g., element A should be a customer of element B).

Rationale: The last, and perhaps most important, part of the software architecture is the rationale which is used to choose between different elements and forms. This is what lends the software architecture its credibility in the early stages of design and development.

Whilst this model provides some insight, it is in and of itself limited.

3.1.1 The chosen Definition of Software Architecture

However, as a complete definition and description of software architecture is beyond the scope of this thesis, the tested and tried version found in the book "Software Architecture in Practice" [3] will be used. The definition is as follows:

> "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

A thorough introduction to this definition is given in Bass et al. [3], but a few points should be introduced here:

1. "the structure or structures of the system [...]":

This implies that a program chiefly consists of several elements, and that the ordering between them are of importance. It also implies that this ordering can be changed as a response to different conditions.

2. "which comprise software elements [...]":

This implies that the actual elements chosen to do the job is part of the software architecture. This again implies that although pieces possibly are exchangeable, choosing one rather than the other is an architectural decision.

3. "the externally visible properties of those elements [...]":

At the same time as one includes every last bit of code into the software architecture, one also abstracts to a sufficient level, looking at the observable effects the different elements of the software architecture have. This allows us to introduce constraints on the different elements, both in terms of interfaces and inter-element relations.

4. "and the relationships among them":
Lastly we return to the relationships among the elements. This definition clearly states that the form of the final product is important, and that the different roles elements assume, and the relationships amongst them, are also part of the software architecture.

3.1.2 Implications of this Definition

When looking at the definition used in Bass et al. [3], some might disagree with their notion of what software architecture is. The fact that the actual elements chosen are part of the software architecture might disagree with notions of a good software architecture. However, although enabling the exchange of modules most likely is a good architectural decision, the module which is chosen is scrutinized, looking for the best match within the constraints given. These constraints can be cost, licensing, performance, etc., but they all imply something regarding the software architecture.

However, only the public parts of the elements are of concern. This implies that the use of pointers most likely touches upon software architecture. The reason for this is that modifications made using this pointer generally are observable. In contrast, temporary instance variables are not architectural since they are not observable

Nonetheless, the complete architecture still comprises most of the information concerning a system, and quickly become too comprehensive to document in full. Thus one needs another approach to documenting the software architecture, an approach that can be found by using various views.

3.1.3 Views of Software Architecture

As the software architecture itself is too comprehensive to document properly, it has been suggested that it should be represented using different views which focus on certain areas of the architecture in isolation. This is much the same as is done in building architecture, where one represents different views of the building for different uses. Plumbers and electricians need a set of drawings telling them where which elements goes (e.g., pipes, power intake, etc.), whilst the seller of the house would need another set.

Examples of views in software architecture are the physical deployment view which illustrates where the software will be running (i.e., on which machines) and the logical view which illustrates the different classes and modules, and certain connections between them.

The "4+1" View Model of Software Architecture

A great example of how software architecture can be documented and presented is introduced in Kruchten [15], and is called *The "4+1" View Model of Software Architecture*.

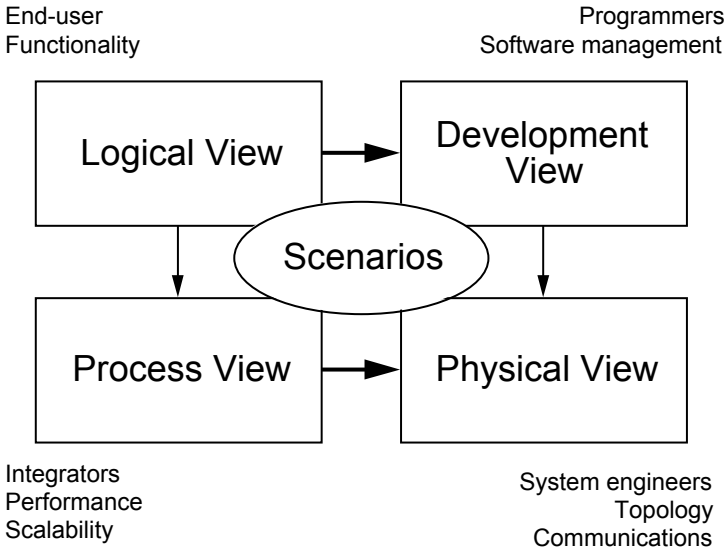

Figure 3.1: An illustration of the five views used in *The "4+1" View Model of Software Architecture*. Image taken from Kruchten [15].

In this model, the idea is that a software architecture can be represented by five views which serve very different purposes.

Four of the five views are related to how the system is implemented and how it looks during runtime. These four views are:

1. The Logical View
2. The Process View
3. The Physical View
4. The Development View

The fifth view is used to illustrate how the four other views are connected and highlight how the architecture as a whole fulfills the needs which are placed on it. For completeness, the last view is:

5. Scenarios/Use Cases

A thorough introduction to how these views interrelate and are used is given in Kruchten [15], but they can be summarized as illustrated in Figure 3.1.

The strength of this model is that all the four "main" views are treated as a complete and distinctive representation of the software architecture. It includes everything which it can based on the constraints of the representation for the respective view, and should therefore also include a rationale as

to why this specific set of choices were made. By doing this, the software architects can be confident that their design fulfills all which is required by the system as a whole, but without having to consider every all the information at the same time.

3.2 Goals of a Software Architecture

The primary reason for studying and using software architecture is to achieve certain goals. These can be tied directly into the requirements given by the customer (e.g., "the system shall have an uptime of 99,99%") or can be desired attributes given by the developing organization (e.g., "it must be simple to add support for a new printer").

In Bass et al. [3] they discuss what they call "quality attributes". These attributes are certain characteristics that the final software should possess to some extent (as given in the requirements). Bass et al. present six of the most common qualities; availability, modifiability, performance, security, testability, and usability.

If a traditional First-Person Shooter (FPS) game is used as an example, these quality attributes can be ordered according to importance. One such ordering might be (1) usability, (2) performance, (3) modifiability, (4) availability, (5) testability, and (6) security. Based on such an ordering, the developing organization can prioritize effort and other resources to achieve the results they wish.

When such orderings, or other constraints, are in place, the software architects can begin designing the software architecture. Accordingly, all the choices the software architects make can be argued for in the rationale using the desired qualities as a guideline.

3.3 Designing a Software Architecture

Designing a successful software architecture is a difficult job. There are many ways of doing this, each with their own strengths and weaknesses. Here we will briefly introduce two approaches; domain-driven design and responsibility-driven design.

3.3.1 Domain-Driven Design

In his excellent book "Domain-Driven Design: Tackling Complexity in the Heart of Software" [10], Evans introduce domain-driven design. Domain-driven design is the process where software is designed by understanding the high-level concepts of the domain you are trying to create a program for. If you are developing an accounting application, the developers should get a basic understanding of how the domain works by discussing it with subject

matter experts (e.g., accountants). Together, they articulate a common and explicit model of the domain, and implement the software according to this model.

In addition to presenting domain-driven design, Evans suggests several ways of keeping the model up to date, keeping it a useful part of the development team's knowledge. Of particular interest are the "Conformist" and "Anticorruption Layer" techniques suggested.

Conformist The conformist technique consists of allowing the software architecture of the program you produce to be dictated fully by the supporting technology in use. If, for example, a third party game engine is used, the conformist approach would likely be the best way to produce correct and working software. It implies that you get a thorough understanding of the existing application (i.e., the game engine), and that all the code you produce follow the model and structure in it.

Anticorruption Layer The anticorruption layer approach is an extreme approach. It implies that you do not wish any influence from any third-party software on your own system. This is achieved by inserting a layer between the third-party software and your own application. On the inside, this layer conforms to the application's structure, and on the outside, this layer conforms to the third-party software's structure. In many ways this is a safe approach which allows third-party software to be integrated into the application without affecting the existing structure. On the other hand, if the layer becomes too complicated or slow, it might not be worth the effort.

In any case, the developing organization should at all times be concious of how they integrate third-party software into their own games. The two presented approaches present the extremes, and for most real-life applications, something in between will be the best solution.

3.3.2 Responsibility-Driven Design

In the article "Object-Oriented Design: A Responsibility-Driven Approach" [30] the authors Wirfs-Brock and Wilkerson present a design method which focuses on the responsibilities of the different entities of a system. They start out by addressing some of the problems with the data-driven design method, most importantly including that it can lead to a violation of encapsulation, since the structure of an object becomes part of it's definition.

They then go on to describe the responsibility-driven approach. In responsibility-driven design, the designers inspect the relationships between the different modules as a client-server relationship. A server is an object that can perform a service (e.g., provides data) and a client is an object that requests services (e.g., requests data).

By designing modules in this way, you do not tie down the actual structure of the different modules or classes in the definition, but instead produce

objects which have clearly defined services which they can perform. If we look at a game engine, each main module can be viewed as a server. The rendering engine, for example, provides the service of generating an image that can be viewed by the end user. The physics engine can calculate object interactions, and the sound engine can output sound.

All these are great examples of object-oriented design, as they provide an encapsulated service, and can, in theory, be exchanged by another module which performs the same services (e.g., by adding a wrapper to the new module).

Chapter 4

Software Architecture in Games

First In this chapter a rationale for the need for software architecture in games will be given. This is then followed by a short introduction to game engines, and the chapter is concluded by a discussion of middleware and the middleware's effect on game development.

4.1 Does Games need a Software Architecture

In Blow [4] a short, but compelling, introduction to the evolution of size and complexity of games is given. As can be seen in Figure 4.1, games started out as rather small and limited pieces of software. However, it can be seen from the figure that there were certain architectural decisions taken, either consciously or sub-consciously, to identify and develop sub-modules, which had their own responsibility.

Then, a couple years later, 3D graphics became a more important part of gaming. Following this, the complexity of the source code increased (Figure 4.2). Now the modules are really starting to stand out. However, it is conceivable that a company still would manage to produce a game without needing a software architecture designed up-front.

Finally, when looking at a 3D game from 2004 (Figure 4.3), one can see that there is no way in which a company can produce such a piece of software without considering software architecture. When also considering that each module consist of between 6 and 40 thousand lines of code [4], the engineering challenge of games become intimidating.

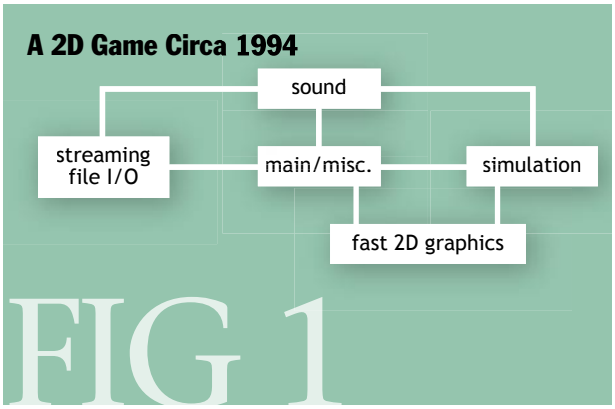

Figure 4.1: An illustration of a typical 2D game from around 1994. Image taken from Blow [4].

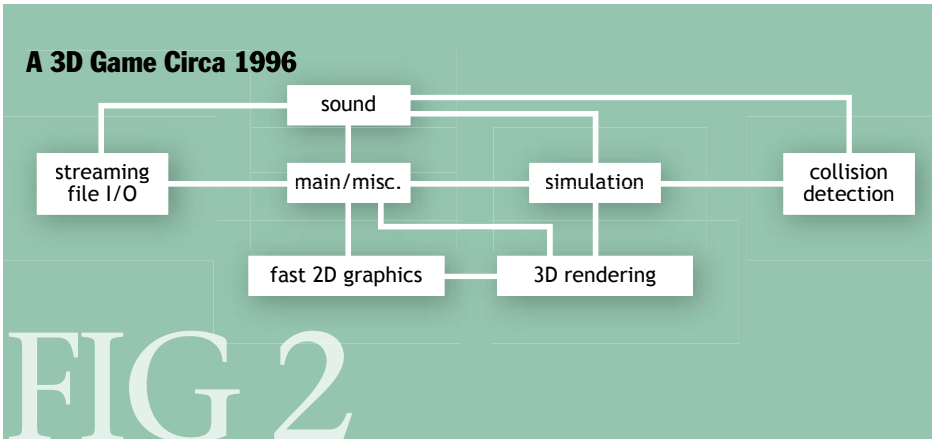

Figure 4.2: An illustration of a typical 3D game from around 1996. Image taken from Blow [4].

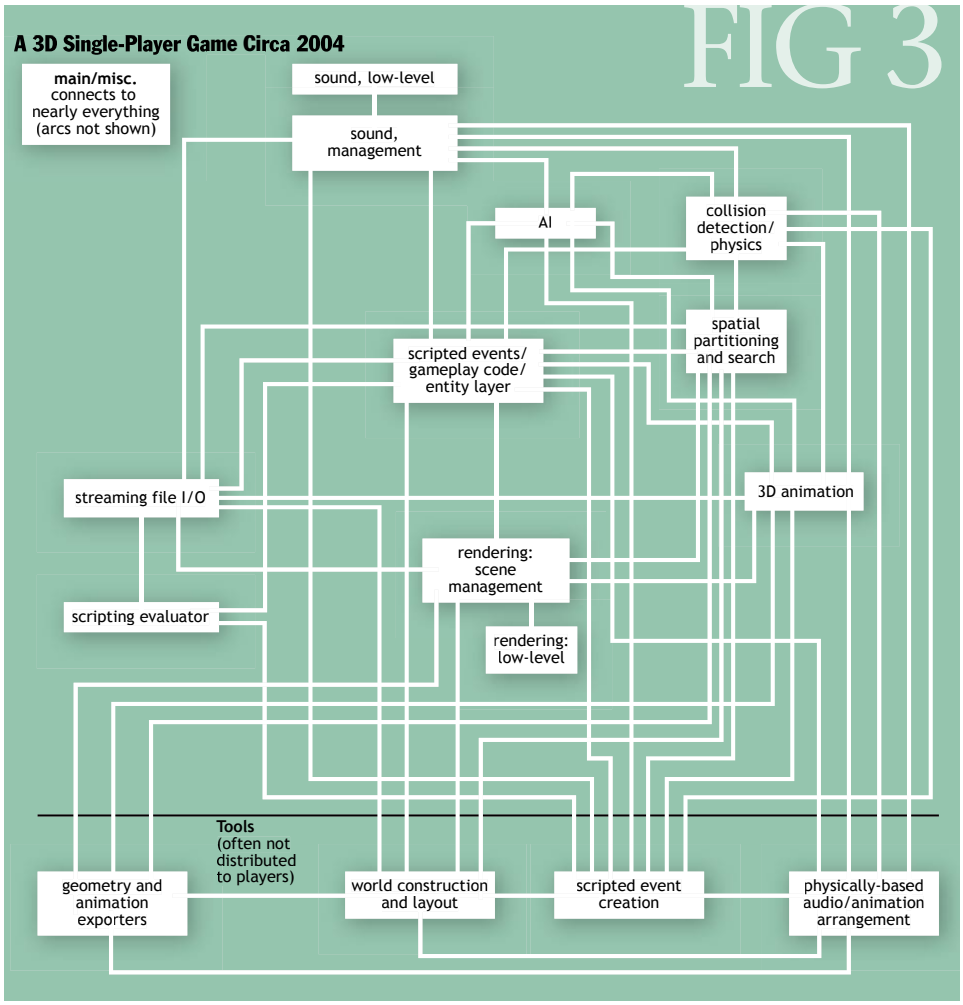

Figure 4.3: An illustration of a typical 3D game from around 2004. Image taken from Blow [4].

Furthermore, when discussing long build times, Blow states that "the best way to avoid long build times is to architect the entire code base to minimize dependencies". Based on this and the increasing complexity of games in software engineering terms, one can conclude that a suitable and well planned software architecture is integral to a successful game[1].

Thus, it has been established that games need to have a concious relationship to software architecture.

A survey performed in Nordmark [18] looks into the relationship that game developers have to software architecture. In the conclusions on this part of the survey, Nordmark state the following:

> "Through this research we have shown that game developers today have a concious relationship to software architecture, and the benefits they reap from using it actively to promote certain characteristics of their software. One of the main findings in the survey is that game developers now rate software architecture as an important part of their game, and that they strive to achieve a software architecture which enables reuse, porting, extension, etc. This hints towards a level of maturity in the game development industry which has been reported and assumed lacking."

4.2 Game Engine Architecture

If one looks at the Figures 4.1, 4.2, and 4.3, one can see that many of the nodes represent general purpose jobs. Examples include sound management, collision detection, and 3D animation. If we take the architecture of a game one step further, these modules can be generalized to such an extent that they can be used in a game engine. This generalization will be discussed in the following sections.

4.2.1 What is a Game Engine

In Gregory [12], the difference between a game and a game engine is defined as follows:

> "Arguably a *data-driven architecture* is what differentiates a game engine from a piece of software that is a game but not an engine. When a game contains hard-coded logic or game rules, or employs special-case code to render specific types of game objects, it becomes difficult or impossible to reuse that software to make a different game. We should probably reserve

[1]The researcher recognizes that smaller (e.g., indie) games can be made in a simpler fashion. However, these developers would most likely also reap the benefits of a proper software architecture.

> the term 'game engine' for software that is extensible and can be used as the foundation for many different games without major modification."

Here certain desirable architectural traits already surface. Gregory has identified a need for the game engine to be data-driven, and that most, if not all, of the game-specific elements should be left out of the game engine. Being data-driven, the game engine should accept new data containing information about the levels, the characters in it, and the possible interactions, and render this to the user in a correct manner. This, in turn, implies that if one uses a complete game engine to create a game, you should only have to focus on the content, not the mechanics allowing you to create a game.

Furthermore, a game engine is usually structured around different modules. These modules each have their own responsibility (e.g., rendering, physics, and lighting), and thus the engine itself is also a responsibility-driven architecture.

4.2.2 Modules used in Today's Game Engines

Guldbrandsen and Storstein [13] performed a study on state-of-the-art game engines, wherein they looked at which modules and middleware were present in the different game engines.

Guldbrandsen and Storstein identify two classes of modules and the modules which are present on virtually all game engines; core modules and gameplay modules.

Below is a short summary of their findings.

Core Modules versus Gameplay Modules

The research Guldbrandsen and Storstein present indicates that the game engines have two main module types; core modules and gameplay modules.

Furthermore they confirm this distinction and present the following statement:

> "One should strive to define what is core functionality, and what is more linked to gameplay. The core should be efficient and stable, and ideally not something a game programmer would need to touch. The gameplay layer should ideally have loose coupling and be extensible for game programmers to incorporate game-specific functionality."

Core Modules

Core modules are characterized by performing a general purpose task like rendering, physics, collision detection, particle system, etc., and are heavy-duty modules which certainly need to be as optimized as possible.

As these modules are developed with performance in mind, making them architecturally pretty is not the main focus.

Gameplay Modules

Gameplay modules are focused on the parts which game developers need to modify to make the specific game they are working on. These consist of entity management, scripting, message passing, etc., and are generally not the most performance intensive modules. Since they are directed at the game developers and designers they are often more "user friendly", i.e., they are developed using functional interfaces, use known architectural patterns, etc.

Below follows a short highlight of two different modules which are typically present in modern games. These are chosen since they affect the job of the creative team. A more thorough introduction to these, and other modules typically present in current game engines, is given in Guldbrandsen and Storstein [13].

Scripted Events

Scripted events are integral to making and fine-tuning the level to match the designers original ideas. Scripts can be used to portray the story, and trigger important events either based on actions, locations, or other indicators.

Most large game engines ship with a scripting language/system like Unreal Engine's UnrealScript [9].

Additional Tools

Most large-scale game engines ship with an assorted set of tools which aid the development of the game. These tools can be everything from level designers, scripting applications, debuggers, etc.

4.2.3 Develop or Buy

Today there are several game engines which can be bought/licensed at a fair price, and they often are considered "mostly-ready game engines" [27]. Examples include UnrealEngine [8], Unity[25], and CryENGINE [7]. And whilst these all show great promise, and can be the correct choice in many situations, a decision to buy a game engine should be carefully considered.

One great benefit is that if the game company decides to buy an existing game engine, they are given a product which is guaranteed to be working, can support bleeding-edge graphics and physics, and allows the company to focus on what makes their game unique, i.e., the content, instead of developing a game engine.

A disadvantage is that the company is limited to what the game engine is capable of doing. Of course, the organization can modify and extend

the game engine, but certain tasks can be too difficult to realize within the bounds of the existing software to be possible.

This and other trade-offs have to be considered when buying a game engine.

Lastly, a few examples of when buying a game engine might be unnecessary are:

- When it is overkill. The game is smaller and can be implemented much simpler than what a game engine implies, or the cost of buying one overshadows the estimated profit from the game.

- When the modifications needed exceed the benefit. This happens when too much of the code base has to change due to requirements within the organization and the game. It might simply be cheaper to build a game, or in-house game engine, from scratch.

- When only one game is planned, or games are peripheral to the company's main focus. If the developing organization only wants to make one game in this genre, or just for promoting their other areas of expertise (e.g., tech-demo of a sound module).

4.3 Middleware

If game can be abstracted into game engines, and the game engines (typically) are structured into modules based on responsibilities, can this be taken one step further?

This occurred to game developers, and thus third party modules, or middleware, entered the stage.

The term "middleware" today seems to be the preferred name for these modules. The term originates from NATO's conference on software engineering [17], and originally was used to illustrate a layer of software which acts as a sort of abstraction on the operating system (see Figure 4.4).

To act as an example, think of a game module which needs to load a certain set of files. A way to simplify porting between platforms, the game developer can create a middleware which abstracts the particulars of file loading, and provides an interface which the application program (game) can use, e.g., `file.load()` and `file.save()`.

If the game is deployed on another platform, or the platform itself changes the implementation, the game developer only have to modify the middleware, and the game will run as normal again.

However, this is the traditional use of the term "middleware". Today, when used in the context of games, middleware is a piece of general software which performs some (rather large) module's work. This can be a physics

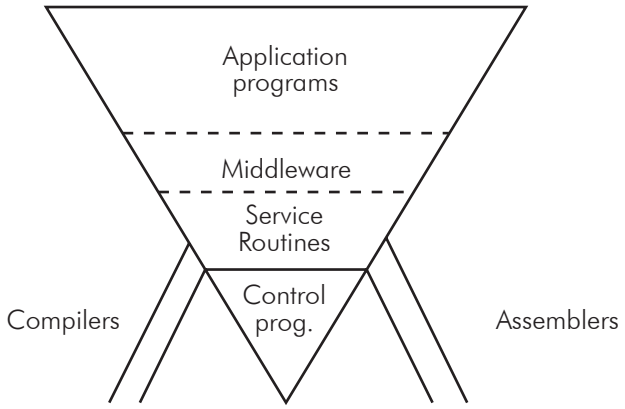

Figure 4.4: d'Agapeyeff's inverted pyramid illustrating middlwares role in a software system. Figure taken from Naur and Randell [17].

engine (e.g., Havok Physics [14]), a sound rendering engine (e.g., FMOD [11]), or any other module of the game[2].

As such, middleware is generally not considered to be a game engine. Middleware can, however, be used as exchangeable modules, focusing on only one aspect of the engine [4]. In general middleware is developed as a stand-alone product, allowing the developing organization to focus entirely on making it a solid piece of software. As discussed in Rollings and Morris [21] this can result in more general, and often better optimized, software than if a general purpose game engine development company should have developed the same functionality.

One can view the emergence of middleware as the next step in generalization of games and game engines. By selecting modules with a clearly defined responsibility like audio or physics, one can create a high-quality, general-purpose piece of software that can be plugged directly into a game or game engine. The engine developers can then choose the most suitable middleware based on purpose, cost, efficiency, etc., and write a small wrapper around it, making it conform to the game engine's existing structure.

This presents an interesting proposition; should you buy a game engine which you have to heavily modify to suit your needs, or do you wish to make most of the game or game engine yourself, and license in parts from other producers? This question should be considered by game developers before they decide to develop or purchase a game engine.

Although the area of game development is an area experiencing rapid and sometimes unpredictable change, it is reasonable to assume that middleware will become more and more prevalent.

[2]Do note that Artificial Intelligence (AI) middleware still has had only limited success.

4.4 Supporting the Creative Team

But how should software architecture be used to support the needs of the creative team?

In Nordmark [18] the second of the four RQs is: "Is software architecture helping game developers doing their job?". The conclusion to this question sums up the findings in a concise manner [18, pages 82 – 83]:

> "From this research, we can clearly see that the game developers reap many benefits from their use of software architecture. From a suitable software architecture, both the creative team and the technical team are able to do their job in a quicker and more efficient way.
>
> If we first look at the creative team, there is one example which illustrates very well how software architecture help them do their job. In the documentary "Beneath the Surface" presented in Section 9.2 they present the possibility to use rapid prototyping when developing their games. This allows the creative team, helped by the technical team, to implement early versions of their ideas quickly, and try them out in-game. If if works well, they can develop the idea further, and if not they can scarp it early, freeing up time to do other things.
>
> Furthermore, the creative team is able to implement levels and characters without having intimate knowledge of programming languages or the inner workings of the game engine. This allows them to focus on their predominant task; developing the content for the game.
>
> A proper software architecture also allows the creative team to request new functionality. If this is a new module, or just an extension to an already existing module, the technical team should be able to consider the actual implementation cost, and if the cost/benefit trade-off favours the benefit, they can implement and integrate it into the game engine without unnecessary workload or ripple-effects.
>
> Another benefit for the technical team is that a sound software architecture which separates generic functionality and game-specific functionality allows for a higher amount of reuse. They can reuse the core components, and replace them if necessary. If we look at the sand engine example in Section 9.1, this shows us that the correct use of software architecture allows the company to extend and modify the capabilities of the game engine without affecting too much surrounding code.
>
> Reuse of components and software architecture allows for quicker development cycles and a higher focus on the unique

parts of the game (i.e., the content)."

As this captures the essence of how software architecture can be used, as discovered in Nordmark [18], there will be no further discussion of the findings here.

Chapter 5

Game Development

In this chapter the relevant particularities of game development will be presented, as well as a short history of game development.

5.1 History of Game Development

This section is based on a similar chapter in the report "Software Architecture in Games" [18], which in turn is based on chapters 16 and 17 in "Game Architecture and Design" by Rollings and Morris [21].

The origins of computer games and the development of these to the home user, started in the 1980s. The four major home computers at the time were the Commodore 64, Sinclair ZX Spectrum, and the Amstrad 464. All of these machines were using slow, 8-bit processors, and with a very limited memory available.

Within these computers there was literally no variance in the available hardware, simplifying the development requirements greatly. The developer also worked directly on these machines. As a consequence of this, there were no risk of the game developed to be too slow, since the programmer constantly verified that the experience was according to the requirements.

Another consequence of the very limited resources was that the developers had to utilize every last bit of memory and every last clock cycle to make the best game they could. Therefore they would drop the Operating System (OS) and all auxiliary functions, and program directly to the hardware available[1].

Since there were no good assemblers or compilers at the time, the developers had to assemble the program themselves. The developer would then write down the op-codes (instructions for the processor) and convert them by hand to hexadecimal digits. These digits could then be entered directly into memory, and then tested. This process was difficult, and since the dif-

[1] This style gave birth to the term "writing directly to the metal"

ferent machine series had different hardware, there was no simple way of porting a game from one platform to another.

After a while, there were released assemblers that were reasonably efficient. This simplified the programmers work quite a bit, removing the need to manually convert the op-codes to hexadecimal. However, there were still several problems. Since the assembler was a software program, it needed a bit of memory to run. This caused a problem, since the games needed every last bit of memory, forcing the games to be assembled in pieces, and then glued together in memory afterwards.

Another problem they had was that they could not debug the program running on the computer. Although mitigated by the limited amount of code, this still was a problem.

After years and years of development, introducing new hardware, OSs, and programming languages, modern day game development was born with the release of Doom, the first game written almost entirely in a high-level language (C).

5.2 Requirements Engineering

In the article Callele et al. [6], a research into how the video game industry performs requirements engineering is presented. In particular, the article looks into how Non-Functional Requirements (NFRs) are specified, focusing on the problems which this can cause in the game development process.

In the introduction to the article, Callele et al. provides a keen observation on some key characteristics of game development organizations [6]:

> "Video games are a special type of multimedia application – an entertainment product that requires active participation by the user. Developed by a multi-disciplinary team, non-functional requirements such as entertaining the user create special demands on the requirements engineering process."

From these circumstances several problems can arise. When considering the requirements of a game, and the diverse team which develops them, Callele et al. state the following [6]:

> "Requirements engineering in the face of such diversity requires the creation of a common (domain) language (and implied world model) specific to the task at hand."

This statement confirms that the game industry can benefit from the use of domain-driven design.

To highlight the problems, Callele et al. perform an analysis of 50 post-mortem reports, categorizing what went right and what went wrong into a set of categories. Based on this analysis, the following conclusion is drawn:

> "Internal factors dominate any other category by a factor of approximately 300%.
>
> Closer inspection of points classified as internal or schedule factors reveals that many, if not most, of the entries are related to classic project management issues."

This again, lead to the conclusion that many problems in game development were caused by "weak management of the transition from preproduction to production". To investigate the validity of this conclusion a case study, which is presented in the same paper [6], was performed. Callele et al. present the following conclusion from the case study:

> "The [case study] showed that, if early versions of preproduction documentation are fed forward to the production team then the production team can provide important feedback to the preproduction team. This communication cycle enables earlier identification of emergent requirements and production constraints and may improve the reliability of the transition from preproduction to production. However, the introduction of production personnel into the preproduction process may have a negative effect on the creativity of the preproduction team."

The key point to take from this article, and which they themselves present, is "that the video game industry could learn a great deal from current research and practice in requirements engineering and project management".

As can be seen in the survey presented in Nordmark [18], agile development methods and rapid prototyping have become more and more popular within the game development industry. This does not invalidate the findings in Callele et al. [6] which were written in 2005, but will affect how game companies arrive at their design, and how this design is documented.

5.3 Evolution of Game Development

In Kvasbø [16] a thorough review of the post-mortem reports in the Game Developer Magazine [24] is presented. This research provided several interesting findings, some of which will be presented on the following pages.

5.3.1 Research Quality

The research performed in Kvasbø [16] is a thorough review of approximately 90 postmortems published in *Game Developer Magazine* [24], looking in particular for features which went right, and features which went wrong.

By classifying the findings into 93 different categories these findings have a far higher resolution than those presented in Callele et al. [6] and thus provides a solid base for statistical analysis.

In the evaluation Kvasbø presents a set of reservations regarding the quality of the quantitative analysis, and the evaluation of how these affect the results. However, when discussing the qualitative analysis done regarding several of the questions, the author is less modest.

Kvasbø's full evaluation of the qualitative analysis is as follows:

> "After being rather negative about many of our statistical findings in the former section, it is now time to look at our confidence in the other side of our findings, the qualitative evaluation of the entire material we have read, not only the parts that were used to create the statistics. As can be seen from the Analysis chapter, we present our findings with a lot more confidence than would be expected from the less than optimal results found via the statistical analysis. The reason for the suboptimal statistical results are outlined in the preceding sections.
>
> What saved us, however, are the reading of the entire material as a whole. By doing this we have gained an insight in the development of games that is very difficult to represent via any quantifiable method. Therefore, we firmly believe that the basis for our analyses in the Analysis chapter are sound and that they reflect actual tendencies in the game development industry."

This implies that the only way to validate the results are by analysing the entire texts themselves, and such a review is beyond the scope of this thesis Because of this, the results presented from the research will be joined by a short evaluation of the basis of the conclusion.

5.3.2 The use of External Game Engines

One hypothesis that Kvasbø investigate is the possibility of increased use of an external game engines. The idea is that more and more game developers now choose to use an external engine instead of developing one themselves.

The statistical results that Kvasbø presents on page 34, as well as the analysis of these on pages 33 – 35, do not confirm or reject this conclusion. Simply, no conclusion can be drawn.

However, in the qualitative analysis presented on page 50, Kvasbø state that "we feel confident that the use of such engines has increased greatly over the years".

This qualitative conclusion arises from the aforementioned review of the entire literature and knowledge of the domain. However, the researcher acknowledges that this conclusion most likely is sound. Indications of this is

presented in Rollings and Morris [21, pages 476 – 478], Gregory [12, pages xiii – xv and 3], and Blow [4, page 32].

5.3.3 Increased use of Middleware

A related issue is discussed in the quantitative analysis of the statistical results. In the section "Development" [16, pages 36 – 38] Kvasbø present a decrease in the number of "what went right" entries related to development categories. Several possible explanations for this is presented, but of particular interest is the increased use of middleware. Kvasbø state the following:

> **"Middleware** The developers uses more and more middleware in the creation of games, and the development phase has therefore become more predictable, thus there are fewer things to report."

Note that Kvasbø present this as a possible explanation, and does not mean to indicate that this is a result of the study. However, if one considers the question "do game developers use more middleware now than earlier", both Rollings and Morris [21] and Nordmark [18, page 80] can be used to support the hypothesis.

It is reasonable that an increased use of middleware reduces overall project complexity since the development organization does not need to develop every last piece of the game engine. This again reduces the number of measures a company needs to take to tackle the development, and thus reduces the number of measures which can be illustrated in a postmortem.

5.3.4 Tools

In the qualitative analysis of the evolution of game development Kvasbø presents interesting sections on different tools. In particular development tools and artistic tools are relevant in the context of this thesis.

Development Tools

The general conclusions, supported by Blow [4], is that there are no perfectly suitable tools for writing games and game engines. A common problem which occur is the long compile and loading times, incurring a heavy penalty on the development team as minor changes and bug fixes takes a lot of time [16, pages 47 – 48]. Kvasbø soundly deduce that this is an issue that has only been affected by the increase in the complexity of the games, not by any revolution in the development tools.

Furthermore Kvasbø state the following [16, page 48]:

> "As this problem cannot be solved by the game developers themselves, it appears from the postmortems that the most usual

> solution is to create workarounds by putting as much of the game specific code that are prone to frequent changes into external files that are loaded at runtime. This is usually mentioned as a success in the postmortems, while it should really be seen as a failure of the development tools developers to supply suitable tools for the game development industry."

Scripting

Kvasbø also discusses the inclusion of a high-level scripting language in games.

Kvasbø notes that the evolution observed through the postmortems illustrates that the level of complexity of the scripting languages have increased. As an example Kvasbø refers to White [28]. In this example the developing organization (Naughty Dog) used their own high-level language to write "practically all of the run-time code" instead of using traditional languages like C++.

This argument is supported by Phelps and Parks [20], in which the authors discuss the benefits and disadvantages of multi-language development. Phelps and Parks identify the use of a scripting language, e.g., to support level creation or AI behaviour, as one of the key areas for multi-language development in games.

The main conclusion to be drawn from the sections regarding tools is that the use of a separation layer between gameplay and core game engine code is beneficial both to the technical and the creative team. This result is confirmed by the study performed in Guldbrandsen and Storstein [13]. The technical team gets shorter build cycles, and the creative team can help develop the gameplay without having intimate knowledge of neither the programming languages or the inner workings of the game engine.

Chapter 6

Web Surveys

Surveys and questionnaires themselves represent great opportunities and great challenges. In this chapter limitations of web surveys, as well as guidelines for creating them, will be presented.

6.1 Challenges with Questionnaire Design

When designing a questionnaire of any kinds, it is important to take into account the different factors which could have an effect on the result. In Vicente and Reis [26] a thorough literature review on the effects on nonresponse in web surveys is performed.

6.1.1 Limitations of Web Surveys

First the authors Vicente and Reis introduce the benefits and disadvantages of web surveys. Anyone designing a questionnaire should go through the limitations, and see how these apply to the population of interest.

Three limitations are identified in the article:

1. Risk of nonresponse
2. Coverage
3. Sampling error

Risk of Nonresponse

The risk of nonresponse is ever present, and the main goal of Vicente and Reis [26] is to illustrate ways in which questionnaire designers can combat this risk. This will be further detailed in Section 6.1.3.

Coverage

As a web survey is only available to those who have an Internet connect, the population sample can exclude certain segments of the population.

Sampling Error

A sampling error is when the selection of the subset of the population is done incorrectly, e.g., due to incomplete coverage.

6.1.2 Types of Nonresponse

In a web survey Vicente and Reis [26] argue that there are three types of measurable rates which are of particular interest to a questionnaire designer; (1) dropout rate, (2) item nonresponse rate, and (3) overall completion rate.

The authors define these as follows:

Dropout rate: the percentage of those who prematurely abandoned the questionnaire, that is represents the percentage of respondents who started the questionnaire but did not complete it.

Item nonresponse rate: the unanswered questions ("no opinions,", "don't knows," or "no answer") as a percentage of the total number of questions in the questionnaire.

Overall completion rate: the percentage of respondents who completed the questionnaire of all those who started the questionnaire.

Note that the first and third are opposites of each other.

6.1.3 Survey Characteristics that Affect Nonresponse

Vicente and Reis [26] then go on to discuss the different categories that affect the nonresponse of a web survey in any way. These are; general structure, length, disclosure of survey progress, visual presentation, interactivity, and question/response format.

A more through description of how these affect the nonresponse is given in Vicente and Reis [26], but a short summary is given here.

General Structure

Vicente and Reis [26] identify two main approaches to distribute the questionnaire; embedded in an e-mail and a link included in an e-mail. The cited research concluded that using an embedded questionnaire increased the completion rate. However, this way of distributing the survey is no longer used due to other benefits of using a web page instead. These benefits include

simpler and better collection of the results (no human interaction is necessary), you can register all the questions a respondent actually answered before abandoning the questionnaire, and that you can actually update the questionnaire if errors are spotted before all respondents have replied without them noticing it.

Vicente and Reis [26] also compare two different visual designs; a scroll design and a screen design. A scroll design can be compared to the traditional paper questionnaire. All the questions are on one page, and the respondent must scroll down the page to view and answer all the questions. At the bottom of the page, the responder can submit the questionnaire using a button.

In a screen design, the questionnaire is divided into smaller parts, each of which consists of only one or a few questions. These are shown on a single web page, and the responder answers all of them before continuing. When continuing, the respondent is show a new screen with a new set of questions, and can continue to answer the questionnaire.

The design choice affects the item nonresponse rate, favouring the screen design (i.e., item nonresponse rate were higher in scroll design than in screen design).

Length

To answer a questionnaire, a respondent has to use a certain amount of her time, and common sense imply that a longer questionnaire will have a lower overall completion rate than a short one. In the literature there is no consensus as to how to measure the length of a questionnaire; should it be the number of questions, the amount of time a respondent is estimated to use, or perhaps something else?

Vicente and Reis [26] cite studies which illustrates this relationship. In addition to the dropout rate, the item nonresponse rate is also higher in a long than a short questionnaire.

Vicente and Reis [26] also discuss the effect of a priori announcement of questionnaire length. The following example will be used:

> There is a set of people which are going to answer a questionnaire. One half of the group is told that the questionnaire will take 10 minutes (collectively called “the short group”), and the other half is told that it will take 30 minutes (“the long group”).

Based on the results presented in Vicente and Reis [26], the following conclusions can be drawn:

- The short group will have a higher percentage of people starting to answer the survey than the long group

- Amongst all those who actually started the questionnaire, the short group will have a higher dropout rate

- If the actual length of the questionnaire is 30 minutes, the short group will have a higher percentage of people starting the questionnaire than the long group, but also a higher dropout rate

Disclosure of Survey Progress

Vicente and Reis [26] also discuss the effect of including a progress indicator in the questionnaire. This progress indicator should show the respondent how much of the questionnaire is completed, as well as how much remains to be done. The authors first identify three different kinds of progress indicators; (1) an indicator which is always visible, (2) an indicator which is shown at key points in the questionnaire, and (3) an indicator which is available on-demand.

For the first indicator (always on), the research performed is not conclusive. Indications exist that such an indicator could actually increase the dropout rate.

Regarding the second indicator (shown at key points), the research is more positive. If this is shown at key points during the questionnaire, it can be used to indicate that "you have already completed half the questionnaire", and can thus be a source of motivation, again increasing the overall completion rate.

Research on the third (on-demand) indicator does not imply any effect on the dropout rate, but shows that such an indicator seldom is used by the respondents.

In addition to research on having an indicator available, research has also been performed on how this indicator behaves. A key conclusion is that any progress indicator needs to be correct and precise. If it does not reflect the respondents feeling of progress, the chance of a dropout increase, especially if the estimates are below the expectations.

Lastly, research on the speed of the progress indication implies that a slow-to-fast (slow progress in the beginning, faster towards the end) made respondents drop out earlier in the questionnaire. This is to be expected. If the respondent feels that she has done a fair amount of work, but the progress bar suggests otherwise, she might consider exiting the questionnaire.

Visual Presentation

The possibilities for graphics and other "eye-candy" on computers can also affect the nonresponse in a web survey. Vicente and Reis [26] presents research which illustrates two key points regarding this:

- A purely textual questionnaire presented with fancy text, colors, etc. resulted in a higher dropout rate than a plain version of the same questionnaire
- The use of logotypes in questions relating to brands or products resulted in a lower item nonresponse rate than plain version of the same questionnaire

Based on this one can conclude that graphics can help on the item nonresponse rates if they are used correctly (e.g., logotypes to enhance visual recognition) and that colorful text and backgrounds should be avoided.

Interactivity

A big advantage of a web survey over a paper-based survey is the possibility to have an interaction with the respondent. This can be used to skip questions which are not relevant for a given responder based on earlier answers, or to modify questions and/or response options based on earlier answers.

Vicente and Reis [26] present research which illustrates some points here:

- The item nonresponse rate is reduced when adding an alert box notifying the respondent that not all questions have been answered.
- The dropout rate increased when employing a forced-response scheme, as compared to a nonforced-response scheme.

Question and Response Format

Vicente and Reis [26] present research which shows that the use of open-ended or difficult-to-answer questions increased the dropout rate, as opposed to using closed-ended questions. This is only natural; if the responder has to put a lot of effort into answering a questionnaire, she might be tempted to quit. However, Vicente and Reis [26] also present research which shows that two questionnaires which contain the same questions, but where one has open-ended and the other has closed-ended answers, the item nonresponse rate were lower in the open-ended version.

Lastly, the authors Vicente and Reis present research which shows that the best format for answering closed-ended questions is the radio button.

6.2 SurveyMonkey

During the research performed in this paper, SurveyMonkey [22] was chosen to create and distribute the questionnaire and the survey.

This commercial online tool allows users to simply create and distribute questionnaires, as well as collecting the responses, enabling analysis of the results.

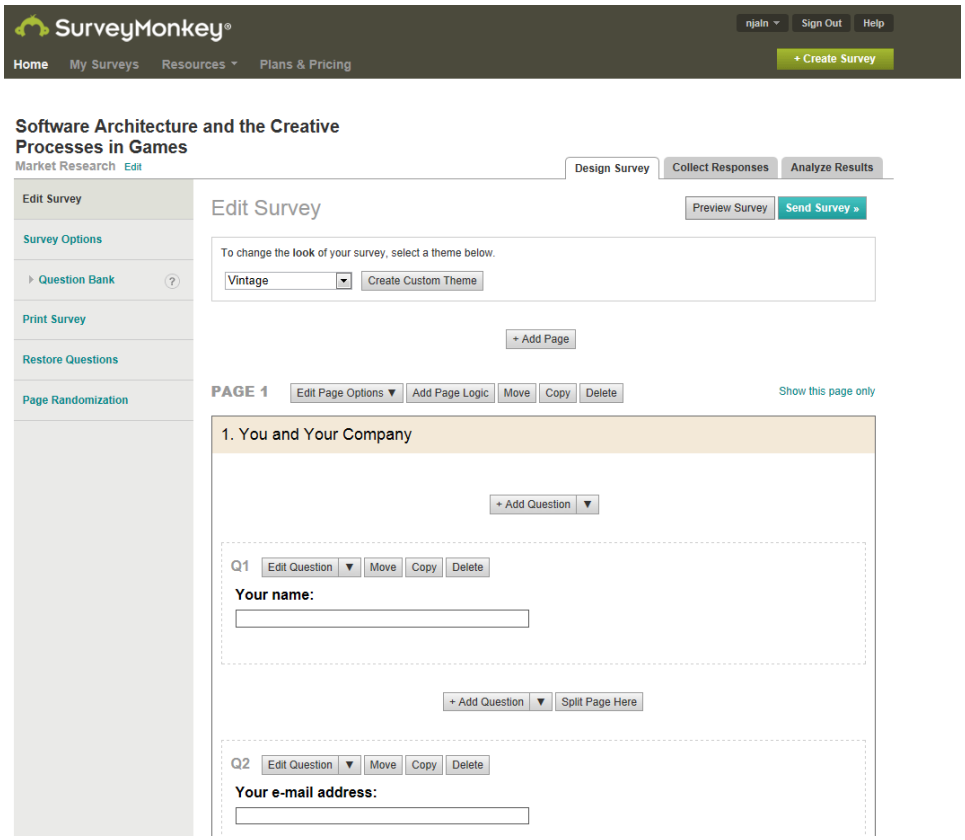

Figure 6.1: A screenshot of the survey designer available from SurveyMonkey.

6.2.1 Designing Web Questionnaires

SurveyMonkey provides a thorough and simple interface to create new web surveys, enabling the user to focus on how to present the questions and on how the respondents can answer the various questions. The designer can specify questions which are mandatory, add skip logic which skips entire pages if they are not relevant, and present the questionnaire in a concise, elegant, and simple way. The questionnaire is simply styled, and the style can be selected by the designer to suit the questionnaire.

A screenshot from the survey designer is presented in Figure 6.1.

6.2.2 Analyzing the Results

After the questionnaire has been distributed to the different respondents and the responses begin to tick in, the questionnaire designer can study and analyse the results in "real time". This means that there is no delay from when the respondent enters their answers, and the designer is able to read

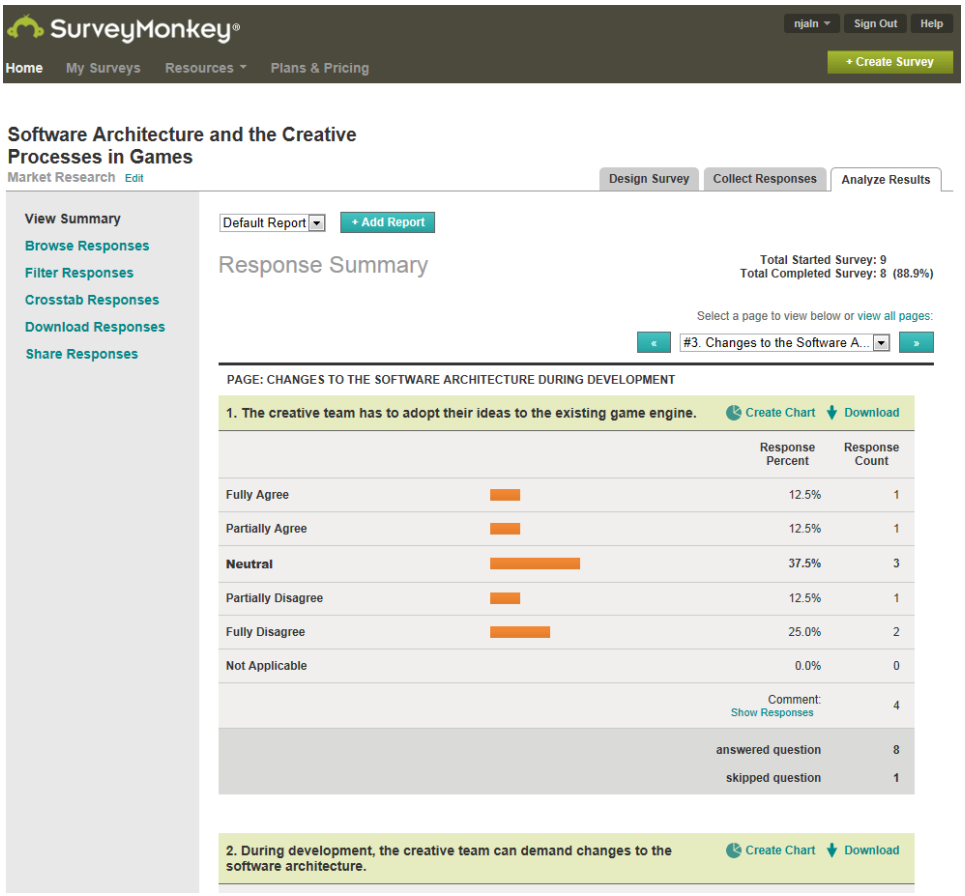

Figure 6.2: A screenshot of the response summary analysis from SurveyMonkey.

them.

SurveyMonkey provides two different means of studying the results. The first is online, using their built-in system for viewing a summary of all responses or one complete response at a time. This way of looking through the answers is great for evaluating if the questionnaire actually ask the questions necessary, and that the respondents interpret them in the same way as the designer meant them.

A screenshot of the online summary of responses is presented in Figure 6.2.

The other way of analyzing the results is to download so-called reports. The reports represent a fixed set of responses which can be downloaded from SurveyMonkey. In these reports, the user can define how the questions shall be represented, the format in which they shall be downloaded, and if the data should be prepared in any particular way. Do note that feature availability is dependent on the type of subscription the user has.

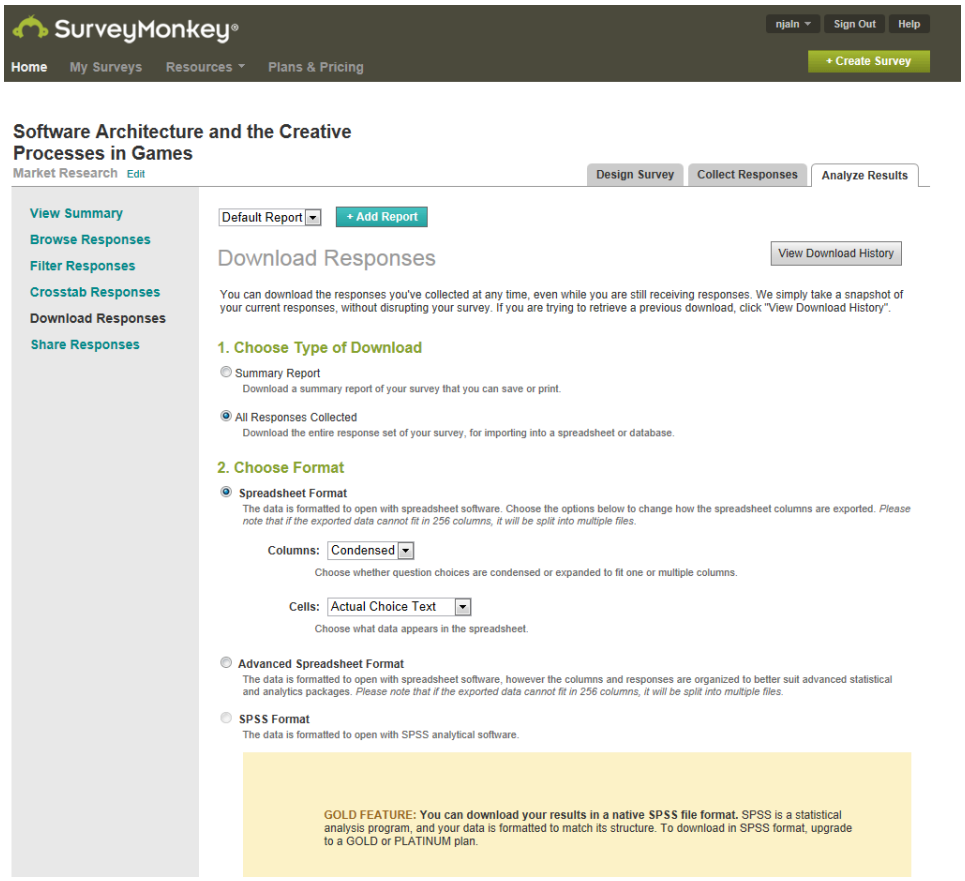

Figure 6.3: A screenshot of the report downloading tool from SurveyMonkey.

A screen shot of the report downloading tool is presented in Figure 6.3

Part III

Research

Chapter 7

Questionnaire

In this chapter the questionnaire sent out to the game developers will be presented.

First the design and rationale behind this design will be presented. After this, the results and their implications will be presented.

7.1 Design of the Web Questionnaire

We will now present how the questionnaire was designed with the research of Vicente and Reis in mind. The full questionnaire can be found in Appendix A.

7.1.1 Limitations of Web Surveys

Firstly, the thoughts regarding the limitations of a web survey, and how these affect this questionnaire, will be presented.

Risk of Nonresponse

The risk of nonresponse is ever present. However, as will be detailed in this chapter, the questionnaire will be designed and distributed in such a way as to minimize this chance.

Coverage

One should note that the population which can answer this questionnaire will need an Internet connection. However, since the respondents are game developers, it is believed that no particular segment of the population will be excluded.

Sampling Error

The sampling error poses the greatest threat to this questionnaire. However, as the questionnaire will be distributed to a large set of game developers, representing organizations of all sizes, the coverage will be decided by the population itself. That is to say, that if only a segment of the population is willing to answer the questionnaire, then only that segment will be represented in the results.

7.1.2 General Structure

The questions for the questionnaire could easily be placed into six categories; You and Your Company, Design of Software Architecture, Changes to the Software Architecture during Development, Supporting the Creative Processes, Changes over Time, and Closing remarks.

Since a screen design is recommended, and the questions lend themselves to this quite easily, screen design was chosen.

7.1.3 Length

This questions for this questionnaire was selected to make the questionnaire as short as possible, but still providing useful and interesting insights.

As a result the questionnaire itself contains 32 questions, including information regarding the company.

During a test, this questionnaire took less than six minutes if the respondent does not enter any comments to the answers and also knows the field. As a result, the researcher felt that announcing the questionnaire time to be five to ten minutes a priori would be the best way to get as many replies as possible.

7.1.4 Disclosure of Survey Progress

Based on the few number of screens (six) it was decided to show the questionnaire progress at the bottom of every page. As the tool used (SurveyMoneky, see Section 6.2) did not easily permit showing the progress only at key points, this was found to be the best choice between not showing it, showing it at the top of the page, and showing it at the bottom of the page.

7.1.5 Visual Presentation

Since the questionnaire was independent of any products or specific items, there was no use for any logotypes or other images. In addition, the questions were generic in form, and did not relate to the respondents interpretation or understanding of particular technologies or concepts which could be explained with a supporting figure. Thus, no graphics were included in the questionnaire.

In terms of colors and other textual "eye-candy", the only thing used was a simple and neutral template for background and font color. This can be seen as being almost identical to a black and white representation in that it does not direct the focus to any particular areas of the questionnaire, but makes it in general less harsh to look at.

7.1.6 Interactivity

In the questionnaire, most questions have to be answered (forced response scheme). The respondents will be notified if they have skipped these questions and will not be able to continue on to the next screen before this has been corrected. According to Vicente and Reis [26] this can lead to a higher dropout rate, but this was still chosen as the approach to get the best results and being able to correlate the different questions based on all replies.

As a positive effect, if a respondent skips a forced question and tries to go on to the next page, she will be notified of the error by text appearing next to the question informing her that it has to be answered. This allows the respondent to immediately see questions which still need answering, allowing her to continue quickly.

7.1.7 Question and Response Format

Most questions have been designed as statements which the respondent should indicate how much she agrees with. These are presented as closed ended questions. However, all the closed ended statement questions also have a field for comments, allowing the respondent to adapt the answers if necessary.

This includes the best of both worlds; respondents are given a set of answering options, and if they feel that it does not perfectly suit them, they can refine the answer in the comment.

7.2 Design of the Paper Questionnaire

In addition to creating a web questionnaire, a paper questionnaire were designed to be used at GDC12 [23].

There were three design constraints for this paper questionnaire:

1. It must have the same questions as the web questionnaire
2. It must include all the information necessary to answer the questionnaire
3. It must fit on one sheet of paper (two-sided)

Given these constraints, the design were simple enough. The result can be seen in Appendix A.

7.3 Analysis

This section begins with an explanation of conventions used in the rest of the chapter.

7.3.1 Question Numbering

In total, the questionnaire consisted of 32 questions or statements which the respondent should answer. Of these, only 21 present questions or statements relating to the research being performed (the others being auxiliary information, e.g., e-mail address). One of these again is a pure comment-questions ("Examples of 3rd party software we use:"). This results in 20 questions in total which will be presented in this section.

These 20 questions are:

Q1: Design of software architecture is an important part of our game development process.

Q2: The main goal of our software architecture is performance.

Q3: Our game concept heavily influences the software architecture.

Q4: The creative team is included in the design of the software architecture.

Q5: Our existing software suite provides features aimed at helping the creative team do their job.

Q6: Our existing software architecture dictates the future game concepts we can develop.

Q7: The creative team has to adopt their ideas to the existing game engine.

Q8: During development, the creative team can demand changes to the software architecture.

Q9: Who decides if change-requests from the creative team are implemented?

Q10: The technical team implements all features requested by the creative team.

Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.

Q12: During development, the creative team has to use the tools and features already available.

Q13: Our game engine supports dynamic loading of new content.

Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.

Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).

Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.

Q17: Today our company uses more 3rd party modules than 3 years ago.

Q18: It is easier to develop games today than it was 5 years ago.

Q19: Middleware is more important to our company today than 3 years ago.

Q20: Game development is more like ordinary software development today than 5 years ago.

To distinguish them from the RQs, they will be numbered with a Q preceding them (Q1 through Q20).

How these questions relate to the RQs can be seen in Figure 7.1.

Figure 7.1: Illustrating which questionnaire questions (numbered Q1 through Q20) relate to the different RQs. In GQM this is equal to metrics and questions, respectively.

As can be seen from Figure 7.1, Q1 does not relate to any RQ. Q1 was included in the questionnaire simply to be able to consider how the respondents view software architecture in terms of their own product.

7.3.2 Presentation

The results from all the questions on the questionnaire will be presented below. They will be presented in tables like the one in Table 7.1, and for ease of reading, only one question will be presented per page.

In all tables, except Table 7.9 detailing Q9, the scale options have been contracted. The contractions are as follows:

Fully Agree	=	FA
Partially Agree	=	PA
Neutral	=	N
Partially Disagree	=	PD
Fully Disagree	=	FD
Not Applicable	=	NA

The complete responses can be seen in Appendix B.

7.4 Questionnaire Results and Analysis

Below the results from the questionnaire will be presented. The analysis will be performed section by section, question by question, not relating them up to the RQs to avoid any bias towards a set of conclusions.

Firstly, however, the results to the question regarding company size will be presented.

Company Size

For the results of this research to be as valid as possible, one should have an understanding of the respondents. To this questionnaire, only one company has more than 10 employees, and this company stated that it had more than 500.

A discussion of result validity is included in Section 11.1.

7.4.1 Design of Software Architecture

Q1 - Design of software architecture is an important part of our game development process.

The statement and results can be seen i Table 7.1

Table 7.1: Answers to question 1

Design of software architecture is an important part of our game development process.

FA	PA	N	PD	FD	NA
7	2	2	1	0	1

This question was asked to get a feel for how important software architecture is to game developers today. Supported by the research by Nordmark [18], the results confirm that software architecture is important. However, the respondents represent mostly smaller companies (up to ten employees).

A reasonable conclusion is that smaller game companies are more adaptive, and that they do not necessarily need to focus on optimizing the different bits and pieces of the game engine as much as possible. This allows the game developers to focus more on how they use software architecture to facilitate and simplify their own work.

This does not imply that larger game developers have a lesser focus on software architecture, but it can be reasonable to assume that they do not have the same priorities. The rendering engine in a top-of-the-range game engine does not necessarily use multiple layers of abstraction, as this can slow it down too much.

Lastly, the comment received from Company B on this question imply the necessity of a, at least general, software architecture: “Oversights in the game software architecture may lead to serious dead ends leading to rewrite of entire systems”.

Q2 - The main goal of our software architecture is performance.

The statement and results can be seen i Table 7.2

Table 7.2: Answers to question 2

The main goal of our software architecture is performance.

FA	PA	N	PD	FD	NA
0	7	2	1	2	1

It is interesting to note that no respondent replied that they fully agreed to this statement. If seen in combination with the analysis of Q1, this can be a result from the respondent chiefly being organizations with no more than ten employees.

However, more than half (7 out of 13) partially agree. Three of the respondents also provided comments to this question. These comments are:

- "Performance plus functionality."
- "Also future change, ability to be datadriven, optimised deployment processes, ease [of] automation/scriptability, testability"
- "Main goals are:

  1 performances

  1.5 memory consumption

  2 actual purpose of the software.

  Real time softwares as games *must* perform according to the platform requirements in order to see the light of the day regardless of the content"

When the comments are considered in conjunction with the seven who partially agreed, one can conclude that performance still is a concern for many game developers. Especially the last comment is really telling; the game must push through the contents it needs to be a game if it should reach the market.

In summary, performance is still a concern, but supporting features is becoming more and more important.

Q3 - Our game concept heavily influences the software architecture.

The statement and results can be seen i Table 7.3

Table 7.3: Answers to question 3

Our game concept heavily influences the software architecture.

FA	PA	N	PD	FD	NA
4	5	1	2	0	1

As can be seen in the results, most game developers agree to this statement. This means that they allow their underlying software to be adapted to suit the needs of the game, making good use of both the game engine and software architecture. However, allowing the game concept to dictate the software architecture may incur some disadvantages.

One respondent provided the following comment: "Entirely depends on the game concept requirements but in general: the more generic, within boundaries, the better". This highlights an important area of game development. Supported by the research by Guldbrandsen and Storstein [13], one can conclude that there should be a separation between generic modules (core) and modules specific for a game (gameplay). This enables reuse of the core components, and still allows the game concept to affect the software, balancing it between reuse (saves money) and suitable game engine (simpler to implement the game).

Q4 - The creative team is included in the design of the software architecture.

The statement and results can be seen i Table 7.4

Table 7.4: Answers to question 4

The creative team is included in the design of the software architecture.

FA	PA	N	PD	FD	NA
5	3	3	1	0	1

Interestingly, as many as eight of the respondents indicate that the creative team to some degree is included in the design of the software architecture. There can be many different ways in which the creative team contribute, and three ways are presented in Nordmark [18]:

(1) Which game to make By deciding which game the company should make, they also lay out the main constraints for the software architecture.

(2) New in-game functionality By requesting new in-game functionality the software architecture might have to be altered.

(3) New development features By requesting new development features, the software architecture might need a large rework.

Furthermore, in smaller game development organizations the employees might need to participate in several ways. The lead artist can also be one of the people adapting or developing the game engine. This enables the creative team and the technical team to blend their roles, and adapt the game engine in a way which in turn is beneficial to the creative team.

One comment which also highlights an important aspect is: “This is mostly true when working on the tools the creative team will be using. It rarely applies to in-game specific features”.

But for whom are these findings relevant?

It is interesting to note that the large respondent (500+ employees) indicated that they were neutral to this statement. This agrees with the reasoning presented above: In a large company one can employ one or a few dedicated architects. These architects receive requests from all the other roles, and suggests a suitable software architecture. However, only one large organization is not enough to conclude on this.

Regarding the smaller game developers one can conclude that they allow the creative team to affect the software architecture in different ways. This can also include directly affecting the design to some extent.

Q5 - Our existing software suite provides features aimed at helping the creative team do their job.

The statement and results can be seen i Table 7.5

Table 7.5: Answers to question 5

Our existing software suite provides features aimed at helping the creative team do their job.

FA	PA	N	PD	FD	NA
8	4	1	0	0	0

Basically one can conclude that the game engines, and the supporting tools, provide features which help the creative team. This is further supported and refined in the comments. This indicates that there is now an increased focus on the creative aspects of game development, and that the software should support these processes.

One comment states that there are "two software tiers, that aims at very different levels of artist integration: Visual Studio and Unity3D". This is further supported by the discussion of Q3. The creative team, and the particular game in development, should preferably only interact with and change the gameplay level of the code. However, there will always be particular features or actions which are not available at such a high level of abstraction, and in these circumstances the creative team, perhaps with support from the technical team, needs to dive into the source code of the game.

Q6 - Our existing software architecture dictates the future game concepts we can develop.

The statement and results can be seen i Table 7.6

Table 7.6: Answers to question 6

Our existing software architecture dictates the future game concepts we can develop.

FA	PA	N	PD	FD	NA
1	1	6	2	3	0

This statement touches upon the very foundation of how a game company looks upon themselves. Do we restrain ourselves to the existing technology, or do we wish to create what our creative team conjures up?

Interestingly, eight are neutral or agreeing with the statement, and five disagree. Based on these numbers, it is impossible to conclude one way or the other, but since six are neutral, this seems to be a difficult question to answer.

A hint as to why this is so can be found in the comments. The three comments to this statement are:

- "We have engines that gives us a great benefit when building new games and we would prefer to continue on same engines, however it doesn't fully dictate the games we will make in future, this is primarily market driven"
- "It may influence, but not dictate whenever possible"
- "It makes it a bit more expensive to go to certain genres, but that's it."

These comments indicate that the influence exerted by the existing software architecture is a direct result of a cost-benefit trade-off. The higher the cost of changing, the more influence the existing software architecture exert on the game concepts.

7.4.2 Changes to the Software Architecture during Development

Q7 - The creative team has to adopt their ideas to the existing game engine.

The statement and results can be seen i Table 7.7

Table 7.7: Answers to question 7

The creative team has to adopt their ideas to the existing game engine.

FA	PA	N	PD	FD	NA
1	3	6	1	2	0

On this question no clear conclusion can be drawn. However, there are two comments which indicate a relationship which can explain the divergence in the responses. These comments are:

- "Most of the time, the creative team is not fully aware of the game engine limitations so it is not their job to make it work by locking the creativity to things known to have been done with the engine before, the people who implements just need to make the ideas work one way or another"
- "Technical realities are always something the creative side has to work around."

These comments indicate that there have to a trade-off between the creative freedom and the technical limitations. It is axiomatic that if an idea which is not supported in the current technology should be implemented, either the idea has to be adopted to the existing technology, the technology adapted to the idea, or something in between. Which one of these is chosen depend on a cost-benefit analysis.

Q8 - During development, the creative team can demand changes to the software architecture.

The statement and results can be seen i Table 7.8

Table 7.8: Answers to question 8

During development, the creative team can demand changes to the software architecture.

FA	PA	N	PD	FD	NA
5	4	4	0	0	0

Before the results from this question is discussed, an excerpt from the e-mail sent to the developers will be presented. The e-mail state that "where questions concern the creative team's effect on the software architecture, this can be indirect effects like requesting a new particle system which leads to a change in the software architecture." See Appendix A for the complete e-mail.

With this in mind, one can conclude that the developing organizations are positive to allowing the creative team to request changes to the software architecture. This is good news for the different creative teams, as they can work more progressively throughout the length of the project. This as opposed to laying out all features and tweaks before coding is begun.

One comment which is relevant here is: "Depends how far in development and how big of a changes, the odds of re-factoring an entire system late in production are close to nil, but the development team keeps an open mind at all times". This, combined with the discussion of cost-benefit in Q7, implies that the developing organizations are inclined to prioritize the wants and needs of the creative team, given that the cost-benefit trade-off is favourable.

This agrees with common sense; if the increased experience of a change is suitable, and the deadline is far enough away, this change will be implemented. On the contrary, if the deadline is close and it is a minor increase in experience and takes a lot of work, it will not be implemented.

Q9 - Who decides if change-requests from the creative team are implemented?

The statement and results can be seen i Table 7.9

Table 7.9: Answers to question 9

Who decides if change-requests from the creative team are implemented?

The Technical Team	Management	The Creative Team
1	4	5

On this question a clear advantage of the digital version over the paper version of the questionnaire was discovered. The goal of the question was to get the respondent to chose which one of these, which all have a say, gets the final word when deciding if a change gets implemented.

On the paper questionnaire, three out of five chose none or more than one response. On the web questionnaire four respondents commented that they would like to choose more than one response. Based on this, and to be able to consider the results, only those who chose one option will be included in the results presented.

When looking at the results, one can see that the creative team is often chosen as the ones which have the last word. This is only natural; if the creative team really needs this one key element to be able to tell the story of the game, they should be the ones to decide. On the other hand, management is also given a lot of weight. A change has a financial impact, and management are the ones who decides if the money should be spent (probably supported by a cost-benefit trade-off).

Q10 - The technical team implements all features requested by the creative team.

The statement and results can be seen i Table 7.10

Table 7.10: Answers to question 10

The technical team implements all features requested by the creative team.

FA	PA	N	PD	FD	NA
4	5	2	1	0	1

This question is a supporting question for Q8 and Q9, and highlights the relationship between the technical and creative team. In general the respondents are positive to implementing these changes.

There are several comments to this question, but one is of particular interest: "[It's] very much a dialogue, we try not to have too formal split between tech and creative team when thinking about this, but prioritise what the user experience should be and when we can ship at target quality".

This comment, when seen in the light of the responses to Q8 and Q9, once again indicates that the decision to implement a feature is a financial decision. If it is financially viable it will be done.

Q11 - It is simple to add new gameplay elements after the core of our game engine has been completed.

The statement and results can be seen i Table 7.11

Table 7.11: Answers to question 11

It is simple to add new gameplay elements after the core of our game engine has been completed.

FA	PA	N	PD	FD	NA
6	3	2	0	0	2

It is interesting to note that the respondents generally are positive. These changes can be fundamental, but, if a suitable software architecture is used, can also be close to trivial.

One interesting comment which shows the downside of such a modifiable software architecture is: "It is simple during prototyping phase, technology-wise. However from a game concept point of view, it is highly dis-recommended and the fact it is simple does not motivate the team to stack up features because the existing one are just not convincing enough".

This should be given some thought. It states that a very modifiable software architecture can actually be a disadvantage. If the creative team does not think that a certain feature is just the way they would like it, they will request a new one even though they would be able to make do with what is currently available. This indicates a need for a proper decision process regarding implementing new features.

Q12 - During development, the creative team has to use the tools and features already available.

The statement and results can be seen i Table 7.12

Table 7.12: Answers to question 12

During development, the creative team has to use the tools and features already available.

FA	PA	N	PD	FD	NA
1	5	2	2	3	0

Here the responses are quite varied, and no single conclusion can be drawn.

However, Company B provides answers and comments which give some insight. Firstly, they provided the following comment on Q8: "Depends how far in development and how big of a changes, the odds of re-factoring an entire system late in production are close to nil, but the development team keeps an open mind at all times". On Q10 they indicated that they partially agree, and supplement with the following comment: "The ones already available and the ones they request along the way".

Given this company's replies, and the results in general on this as well as Q8 and Q10, the following hypothesis is presented:

> In some companies the availability of tools and other software is decided up-front and is a rigid decision. In general, however, adding new tools or software to the existing software suite is more a question of cost than of principle. If it is financially viable, and there exists an actual need, tools can be bought or developed and thereafter used.

Note that this is a hypothesis and not a conclusion.

7.4.3 Supporting the Creative Processes

Three of the four questions in this section relate to specific functionality that a game engine can provide to support the creative team's work. These were asked to get a glimpse into how game engines are used today.

Q13 - Our game engine supports dynamic loading of new content.

The statement and results can be seen i Table 7.13

Table 7.13: Answers to question 13

Our game engine supports dynamic loading of new content.

FA	PA	N	PD	FD	NA
8	4	1	0	0	0

Based on the responses, one can conclude that game engines today support dynamic loading of content. There is one comment which states that any limitations stem from the need to prepare certain content in a particular way.

Another limiting factor can be the different "content" to be loaded. If the game suddenly should support a new type of game mechanic, this might not be as simple as plug and play, but might require changes to the gameplay layer of the game engine.

Q14 - Our game engine has a scripting system the creative team can use to try out and implement new ideas.

The statement and results can be seen i Table 7.14

Table 7.14: Answers to question 14

Our game engine has a scripting system the creative team can use to try out and implement new ideas.

FA	PA	N	PD	FD	NA
6	3	2	1	1	0

Based on the responses, one can see that a scripting language in game engines (as discussed on Page 40 in Section 5.3.4) is becoming prevalent. This simplifies the development and allows shorter turn-around time from a feature request to it has been implemented and tested.

Although this questionnaire does not enable such a conclusion, one can assume that those who disagree might have made their own game engine, and have not prioritized the effort needed to implement such a language.

Q15 - The creative team is included in our development feedback loop (e.g., scrum meetings).

The statement and results can be seen i Table 7.15

Table 7.15: Answers to question 15

The creative team is included in our development feedback loop (e.g., scrum meetings).

FA	PA	N	PD	FD	NA
11	0	1	0	0	1

The results here show that the game company appreciates the feedback from the creative team in their development. This allows the technical team to continuously work towards the goals of the creative team, and constantly receive feedback on their work.

This implies that the organizations work closely across the different disciplines the employees represent, overcoming one of the perceived barriers of game development.

Q16 - Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.

The statement and results can be seen i Table 7.16

Table 7.16: Answers to question 16

Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.

FA	PA	N	PD	FD	NA
9	2	1	0	0	1

As most respondents agree with this statement, one can observe a tendency towards rapid prototyping. This way of working has been identified in Nordmark [18], and allows bad ideas to be discarded before too much work has been put into it. Allowing the creative team to do this job themselves, while the technical team can focus on other tasks allows the organization to spend a minimum of money on testing out new ideas, whilst still producing new, high-quality aspects of their games.

7.4.4 Changes over Time

In this section of the questionnaire, the questions relate to how game development has changed over the years.

As the questions can be quite self-explanatory, all unnecessary discussion will be avoided.

Q17 - Today our company uses more 3rd party modules than 3 years ago.

The statement and results can be seen i Table 7.17

Table 7.17: Answers to question 17

Today our company uses more 3rd party modules than 3 years ago.

FA	PA	N	PD	FD	NA
6	0	2	0	1	3

From the results one can conclude that third party software is more prevalent today than before. This confirms predictions made by several authors [18][21][13]; buying a good middleware will provide a better result than what the organization themselves can produce at the same prize.

Q18 - It is easier to develop games today than it was 5 years ago.

The statement and results can be seen i Table 7.18

Table 7.18: Answers to question 18

It is easier to develop games today than it was 5 years ago.

FA	PA	N	PD	FD	NA
7	3	1	1	1	0

The respondents tend to agree, but one comment should be highlighted:

> "The challenges have changed and the quality bar has risen, it is more accessible to people less interested in nerdy things nowadays (engines like Unity reduced/removed the low-level aspect of the development), but developing a great game is still as challenging as before, the problems to solve just have evolved."

This is an interesting point. Technically it is simpler, but making a great game consists of a lot more than just allowing input to dictate movements on screen. The concept of a great game is unique, and the challenge of making one is no different than before.

Q19 - Middleware is more important to our company today than 3 years ago.

The statement and results can be seen i Table 7.19

Table 7.19: Answers to question 19

Middleware is more important to our company today than 3 years ago.

FA	PA	N	PD	FD	NA
3	4	2	0	2	2

This question complements Q17, and the responses look similar. The conclusion must be that middleware plays an important role in todays games and game engines.

Q20 - Game development is more like ordinary software development today than 5 years ago.

The statement and results can be seen i Table 7.20

Table 7.20: Answers to question 20

Game development is more like ordinary software development today than 5 years ago.

FA	PA	N	PD	FD	NA
0	5	3	1	4	0

This question presents some interesting findings, mostly in the comments:

- "Nope. It was software development then, and still is now"
- "Game development requires a more eccentric creative problem solving than development in most of other industries and this will probably remain true forever"
- "I think that the tools available today moves game dev further away from 'ordinary software dev' "

Chapter 8

Survey

In this chapter the follow-up survey sent out to game developers will be presented.

First the rationale behind the design as well as the design will be presented, directly followed by a presentation of the results from the survey.

8.1 Survey Design

From the questionnaire presented in Chapter 7, a new set of questions arose. In this section these questions and the rationale behind them will be presented.

Note that the questions in the survey are numbered from S1 through S9. This is to distinguish them from both the RQs (numbered RQ1 through RQ5) and the questionnaire questions (numbered Q1 through Q20).

8.1.1 Introduction

The first section in the questionnaire was used to try and collect information regarding the respondent. This information could then be used to connect the replies between the questionnaire and the survey.

At first, this section was mandatory. However, the researcher received critical feedback from one respondent. Based on this feedback, the first page was made voluntary, and the following text were placed on the top of the page:

> "All answers on this first page is voluntary, and will only serve to correlate answers between the first questionnaire and this follow-up questionnaire.
>
> All answers given to both questionnaires will be anonymized before publication and will not be used commercially."

8.1.2 Game Engine

The second section of the survey looks into how game engines are used amongst the respondents. This should give some insight into how prevalent external game engines are, as well as the use of middleware.

S1 - Which game engine do you use?

This question should provide an insight into how many develop their own game engines and how many use external game engines.

In addition, it could be used to see if there are particular game engines which are more popular than others amongst the small game developers (up to and including ten employees).

S2 - Did you use any middleware or third party modules in your game?

Furthermore the survey looks into the use of middleware. Instead of asking general questions regarding which type of middleware they use, the question is phrased so as to get a look at which middleware is the most popular.

On the first questionnaire the respondents were asked for examples of third party software that they used. This information will be included in the analysis of this question as it is basically the same question.

S3 - Do you have any thoughts regarding the evolution of game engines in the future?

Lastly in the game engine section the respondents were given the opportunity to provide some of their insight into how game engines will evolve. The question was included in order to get as much information as possible from the respondents regarding a few themes:

- What types of game engine will exist
- How will game engines be developed
- How will game engines be used

However, as to not guide the respondents answers, this question was phrased as openly as possible.

8.1.3 Software Architecture and the Creative Team

The third section of the survey looks into how the creative team affects the software architecture, and if the creative team uses features which is enabled by a suitable software architecture.

S4 - In what ways are the creative team allowed to contribute to the design of the software architecture?

Q4 in the questionnaire (see Page 61) asked whether the creative team is included in the design of software architecture. Since the results there did not provide much insight into how this was done, this question will elaborate on this.

S5 - Which features in particular do your software suite provide to help the creative team do their job?

Q5 in the questionnaire (see Page 62) asks whether the existing software suite (game engine, supporting tools, or other software) provides features which help the creative team do their job. This question tries to identify the particular features which are present.

S6 - To what extent do the creative team use the game engine and its features to try out new ideas?

Expanding Q14 (see Page 71), this question looks into which parts of the game engine the creative team actually uses when designing new aspects of the game.

S7 - Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?

This question tries to shed light over how features provided by a good software architecture are included in the creative team's progress.

8.1.4 Implementing Changes

On the final page of the survey, the respondents were given an example as context for the last two questions. The example was as follows:

> "You have a game engine which supports real-time physics interaction between the game world and the entities present in it (and between themselves). The game world is imported with a certain physical, constant appearance (e.g., rocks and other terrain) which is used for calculating the physics interaction. This appearance is static.
>
> The creative team then requests a new feature, in which they would be able to introduce an earth-quake which would need to alter the physical appearance of the game world. As a consequence of this change, the system for loading the game world would need to be made dynamic, and also support real-time altering of the physical appearance of the game world."

S8 - How would your company reason about implementing the above mentioned change?

This question touches upon Q8, Q9, and Q10 and to some extent also Q11. The objective of the question is to get a glimpse of how the decision process regarding changes are handled, as well as how the company would progress with the implementation. The answers will of course vary wildly between the respondents, but can support a conclusion to the RQs.

S9 - Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?

This is merely an extension of S8, but also extends directly on Q9 (see Page 66) and needs no further discussion.

8.1.5 Relation between the Survey and the RQs

In Figure 8.1 the relation between the questions in the survey (called metrics in GQM) and the RQs (called questions in GQM) is presented.

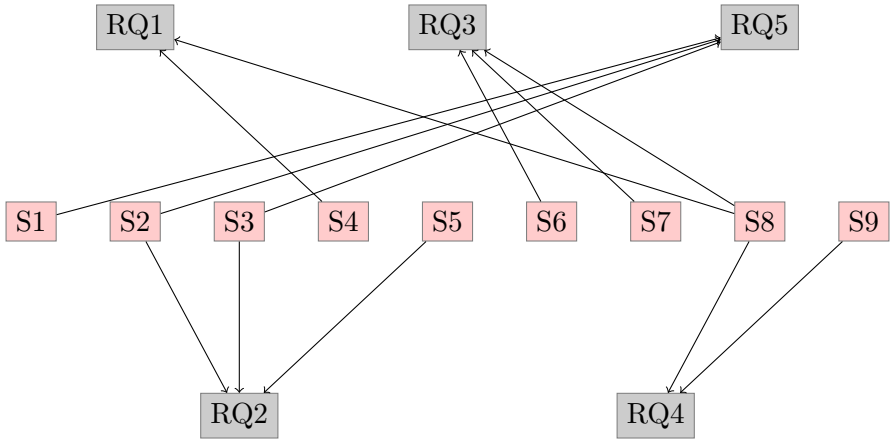

Figure 8.1: Illustrating which survey questions (numbered S1 through S9) relate to the different RQs. In GQM this is equal to metrics and questions, respectively.

8.2 Analysis of Responses

Below, a summary and an analysis of the six responses to the survey will be presented. For the complete responses, please refer to Appendix D.

8.2.1 Game Engine

S1 - Which game engine do you use?

Four of the six respondents replied that they used external game engines, whilst the two last replied "custom" and "our own".

This confirms the hypothesis that external game engines have become more and more prevalent.

S2 - Did you use any middleware or third party modules in your game?

Four of six respondents replied that they used middleware in their game engines. When reading through the responses, one in particular stands out:

> "On the previous game, developed with the in-house engine, FMOD was the only third-party software licensed and used for audio playback. SDL was also used for the Mac OS X port but was a free open source library. On the next projects, done with Unity 3D, we obviously use Unity as middleware. Because of the way their engine is constructed, it is highly unlikely we will need any additional third party modules as FMOD, Beast lightmapping and Umbra occlusion culling solution are integrated to the engine and part of the engine license."

In addition to specifying which middleware the organization has used, it also points at evolution of game engines. Based on this reply, one clear trend of game engines shows itself; middleware becomes more and more prevalent, even in licensable game engines.

From the responses to both this question (S2) and the examples of third party software from the questionnaire, the following list of used middleware and third party software can be compiled:

- Autodesk Beast - Middleware for lighting
- Autodesk Scaleform - Middleware for user interfaces in games
- Away3D - Real-time 3D engine for Flash and ActionScript 3.0
- Bink - Video codec for games
- Box2D - 2D physics engine
- DirectX - Windows' collection of APIs to handle multimedia
- Flash - Multimedia platform for the web
- FMOD - Middleware for audio

- libvorbis - Audio codec
- NVIDIA PhysX - Physics engine
- SpeedTree - Middleware for trees
- Substance - Texture designer
- Umbra - Middleware for rendering optimization
- Unity - A 3D game engine
- UnrealEngine 3 - A 3D game engine

S3 - Do you have any thoughts regarding the evolution of game engines in the future?

This question received three high-quality answers. The following key points should be noted from the replies:

Multi-platform: The ability to create a game once and build it to run on different platforms allows game developers to reach a much larger audience, and at the same time being able to focus on the work of creating the game without always considering porting.

Quality of features: Whilst most game engines today present new features, the replies, and the replies to S2 above, indicate that the quality of the feature is more important than the quantity. If a really impressive, bleeding-edge feature is included in the game engine, most games will not use it until it works properly and is simple to use.

Simplicity: The usability of a game engine has increased rapidly from the earliest game engines to those who dominate the market today. The replies indicate that this trend will only continue, and that game engines which are difficult to use will fall behind in the competition. However, this simplicity must not be at the expense of freedom. As there are limits to how much freedom a point-and-click interface can provide, the companies should still be able to edit the source code, allowing them to develop new and novel features.

Completeness: A game engine today must present more than "just" a rendering engine which accepts input data, and produces a game. The game engine needs to have a host of supporting features and tools, relieving the individual organizations of the run of the mill development tasks like taking models from modelling tools and converting them to game engine-compatible data formats, or handling save games.

The researcher expects that game engines will continue their evolution towards the characteristics above.

8.2.2 Software Architecture and the Creative Team

S4 - In what ways are the creative team allowed to contribute to the design of the software architecture?

There are two recurring themes in the responses. Firstly, the creative team affects the software architecture indirectly through working with the technical team. Secondly, the main areas they affect relate to how tools interact with the game. This can be a result of discussions regarding workflow issues, or based on the functional needs of the creative team.

Thus, the creative team does not affect the software architecture directly, but through requests made to the technical team.

S5 - Which features in particular do your software suite provide to help the creative team do their job?

When going through the replies, it becomes immediately obvious that all companies both have and desire functionality which lets the creative team directly import new assets and try them out in-game. By enabling the creative team to directly import their new assets from the tools into the game, without changing a single line of code makes rapid prototyping possible. Rapid prototyping, as discussed in this thesis as well as Nordmark [18], allows the creative team to simply test out if a new idea is viable, and discard it if it is not. This results in better use of the creative team's time, and, in the end, streamlines this area of game development.

S6 - To what extent do the creative team use the game engine and its features to try out new ideas?

Quite a few of the respondents have already touched upon this question in S5. The responses further confirm the preferred workflow with regards to rapid prototyping. One statement which deserves notice is: "Naturally, the more data and tools driven we can be the more the creative team can [experiment]".

In the extension of this, another reply also identifies the automatic transition from tools to game as an important aspect. If the creative team simply can test out their new ideas, they will try them out often, and produce a better game.

Additionally, if the creative team possesses some light programming skill, they would also be able to take a copy of the project and try out altering the source code on their own. This allows for a more fundamental approach to implementing new features, but can still be feasible for certain organizations.

S7 - Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?

In general, the responses to this question confirms and supplements those presented in S5 and S6.

However, one respondent introduced an interesting way of work using "feature-oriented teams". In the example given, these team consist of one coder (technical team), one artist, and one designer (both from the creative team). This allows them to focus on particular features are represented as a single unit, allowing work to progress quickly without having to wait for any external resources.

8.2.3 Implementing Changes

For simplicity of reading, the example the following two questions relate to will be repeated here:

> "You have a game engine which supports real-time physics interaction between the game world and the entities present in it (and between themselves). The game world is imported with a certain physical, constant appearance (e.g., rocks and other terrain) which is used for calculating the physics interaction. This appearance is static.
>
> The creative team then requests a new feature, in which they would be able to introduce an earth-quake which would need to alter the physical appearance of the game world. As a consequence of this change, the system for loading the game world would need to be made dynamic, and also support real-time altering of the physical appearance of the game world."

S8 - How would your company reason about implementing the above mentioned change?

In addition to providing an interesting insight into how different game developers consider changes of this magnitude, there are a few recurring themes in the replies.

Initially there is a decision process consisting of two main considerations;

Firstly, how important is this feature for the experience of the user? How much better will the game be with this feature? Conversely, how much will be lost if it is not implemented?

Secondly, if this feature will be implemented, how much will it cost in terms of time and money? Will the added workload be justified?

If this consideration is favourable, the organization will start considering how the feature should be implemented. From the answers, one can see that this is started with a discussion between the creative team and the technical

team. Here the initial goal, as seen from the creative team, is subjected to technical considerations. Based on this feedback and feedforward, the creative team end up with a specification of the feature. Based on this specification, the technical team produce a prototype. Glaring oversights, or new, important elements are added. When both the creative and the technical team is happy with the prototype, it is fixed into production quality code.

S9 - Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?

There are a few interesting conclusions one can draw from the responses.

Firstly, management has the final say if the change significantly alters budget or time estimates. This is not to say that this is done without involvement from either the creative team or the technical team, but in the end, management decides.

Secondly, in the companies which replied, all three groups seem to be treated equally; the technical team, the creative team, and management all get to participate in decisions like this. This makes sense, as these three groups have three different responsibilities. Management should get the game launched on time and budget, the creative team should produce a game which is fun or involving, and the technical team should enable the technology to drive the creative team's content through in a reliable way.

Chapter 9

Experiences

In this chapter a short presentation of the experiences gained from communicating with game developers will be given.

The presentation will focus on previous knowledge, the questionnaire, and the survey. In particular, challenges which arise when communication with game developers will be discussed.

9.1 Previous Experience with Game Developers

The researcher gained experience in communicating with game developers during the study "Software Architecture in Games " [18], and the first thing one should consider when sending any form of request to a game developer, is that you are asking them to spend time from their already tight schedule on you.

Thus, when authoring an e-mail, a questionnaire, or a survey for game developers, one should prioritize making the text concise, expressive and unambiguous. The shorter and more direct the text is, the better. Even though this is practiced, there are no guarantee that the game developers will answer, and in general there will be a low overall completion rate.

9.2 Questionnaire Experiences

9.2.1 Design

When designing the questionnaire, the questions were carefully worded as to convey their meaning as unambiguously as possible. Still, the researcher experienced that some of the questions were interpreted differently than they were intended. However, the replies still provided a solid foundation for concluding on the five RQs.

9.2.2 Distribution

The questionnaire was distributed to game developers in two different ways; as a paper questionnaire at GDC12 and as a web questionnaire distributed by e-mail.

On the paper questionnaire (see Appendix A), the questions and the necessary information to answer them were presented on one sheet of paper. This made the questionnaire less overwhelming for the respondents. However, the questionnaire was distributed on the show floor at GDC12, and it proved difficult to get respondents, as they were constantly busy promoting their own products, or trying out the products of others.

When distributing the web questionnaire, an e-mail was authored which contained information regarding the research, and the information necessary to answer the questionnaire. This e-mail can be seen in Appendix A.

However, despite both the e-mail being concise and the questionnaire to take less than ten minutes to answer, only 8 of 40 unique recipients answered the questionnaire. This confirms the researchers experience from Nordmark [18]; the overall completion rate will be low.

9.2.3 Collection

As presented in the previous section, only 8 of 40 recipients responded to the web questionnaire, resulting in a overall completion rate of 20%.

However, the tool used (SurveyMonkey) proved to be a valuable asset. It easily enabled downloading the responses in spreadsheet format, and this again enabled simple analysis of the data.

The researcher received some feedback on e-mail on the questionnaire and the tool used (SurveyMonkey). This feedback was positive, and supported the decision to continue to use SurveyMonkey throughout the research.

9.3 Survey

9.3.1 Design

When designing the survey, the responses to the questionnaire served as a basis for identifying areas which would benefit from further research. Once again SurveyMonkey was chosen as the tools to create and collect responses, and this worked well.

As on the questionnaire, the wording was chosen carefully to avoid ambiguities, and this time the researcher did not experience any significant mismatch between the intended meaning of the questions and the responses.

9.3.2 Feedback

The first section of this survey was meant to allow combining the responses from the questionnaire and the survey. However, the researcher received feedback from one anonymous respondent which indicated that he or she was afraid that the data collected would be exploited for financial gain. Based on this feedback, the researcher updated the introductory text and allowed the respondents to skip these questions.

No other respondent, before or after this change, chose to refrain from providing this information. However, the researcher recommends that providing identifying information should be optional, as this likely will prevent such feedback.

9.3.3 Collection

Since all recipients of this follow-up survey previously had agreed to answering the questions, the response rate was significantly higher this time around. 66% answered at least parts of the survey (six out of nine), and 55% completed the survey (five out of nine).

9.4 Summary

The researcher would recommend future research into game development to be performed in similar ways.

Firstly, one should perform a short pre-study, only highlighting the main areas of the research. In this pre-study, the respondents should be asked whether or not they are willing to answer some follow-up questions. In general, respondents which answer yes to such a question will be more inclined to use their time and answer the questions thoroughly.

Based on this pre-study, a more thorough and targeted survey can be designed and sent out to recipients who are willing to participate in the research. This should increase the number of high-quality answers, and allow for a better research to be performed.

Chapter 10

Evaluation

In this chapter an evaluation of the thesis will be given. In particular, the research method, the actual research performed, and the thesis' strengths and weaknesses will be discussed.

10.1 Research Method

The research performed in this thesis is a qualitative study, and conforms well to the empirical method described by Basili [1].

The first part of the research was a literature review, investigating different aspects which are related to this research. The literature review provided a good foundation on which the questionnaire could be designed.

The questions for the questionnaire was designed using the GQM approach. This resulted in a questionnaire with 20 questions, relating to the five RQs. The questionnaire was was used to get an understanding of the current practices amongst game developers.

As expected, the responses to the questionnaire also gave rise to new questions. These were formulated as open-ended questions, and were distributed to a set of game developers as a follow-up survey.

The GQM approach has been of great use. This approach lends itself well to creating both questionnaires and surveys, and thinking in terms of goals, questions, and metrics has enabled the researcher to maintain the focus on the overarching goal.

10.2 Research Performed

In this thesis a research is performed which investigates the relationship between software architecture, the creative team, and the development processes.

The initial phase of this research was a literature review which gave the researcher a basis on which the questionnaire could be designed. This

questionnaire was estimated to take between five and ten minutes to complete, and was distributed in two different ways; as a paper questionnaire on GDC12 and a web questionnaire sent out by e-mail.

The effort needed to gather the replies, favoured using the web questionnaire as much as possible. This web questionnaire can then be sent out to a large amount of game developers using a template e-mail. When collecting responses to a paper questionnaire one has to introduce oneself and explain the purpose of the questionnaire over and over again, and this takes a lot of time and effort. With regards to analysis of the responses, the web survey also proved superior. Digital collection has several advantages, e.g., no misinterpretation of handwriting, less time needed to analyze each response, etc.

The feedback on the questionnaire was positive, and the responses proved to be a great asset in both supporting the conclusions on the five RQs as well as designing the follow-up survey. This survey was distributed exclusively as a web survey, and resulted in several high-quality responses which further supported the conclusions.

In summary, the answers two both the questionnaire and the survey as well as the literature review, enabled a conclusion to be reached on the five RQs.

Thus, this research has successfully investigated the relationship between the creative team, software architecture, and the development and decision processes, and provides a valuable insight into how a small game company enable the creative team to work as efficiently as possible.

10.3 Strengths and Weaknesses

A great strength of this research is that it explores a previously little studied area of game development, i.e., the relationship between software architecture and the creative team. The literature review, the questionnaire, and the survey all contribute to the final conclusions, and as such provides a solid foundation for the conclusions drawn within the limits of the validity presented in Section 11.1.

On the other hand, a weakness, or limitation, of this research is that it cannot be generalized to be valid for all game companies, not even the smaller game companies with ten or fewer employees.

However, the research provides an interesting glimpse into the game development industry, and is as such considered successful by the researcher.

Part IV

Conclusions

Chapter 11

Research Conclusions

In this chapter the conclusions on the five research questions will be presented, as well as the validity of the results.

11.1 Validity of Results

The research performed and presented in this thesis has been based on voluntary responses from primarily smaller game developers with ten or less employees. In total, there are thirteen respondents, and from these thirteen, six replied to both the questionnaire and the follow-up survey.

The number of respondents, and the way these were selected (voluntary responses), do not allow the results to be generalized to be valid for the entire population of smaller game developers (ten or less employees), even less to the population of all game developers.

Still, the research is well suited to give a picture of the situation for smaller game developers in the industry today. In addition, it has some strong qualitative properties as several of the responses received, both on the questionnaire and the survey, are both detailed and of a high quality.

11.2 Research Question 1

The first RQ is:

> What are the primary ways in which the creative team can affect the software architecture in a game?

From the literature review it is known that there are three ways in which the creative team can affect the software architecture; (1) deciding which game to make, (2) requesting new in-game functionality, and (3) requesting new development features.

This provides an answer to the RQ, but the research performed in this thesis goes into more detail.

The research confirms the conclusion that the creative team affects the software architecture in a game. Based on the questionnaire it is concluded that the game, as defined by the creative team, does have a certain effect on the software architecture, and, in addition, that the organization can discard an old software solution if it is necessary to make a new type of game. As it is the creative team who develops ideas into new game concepts, it is definitively given a lot of power over the main constraints of the software architecture.

However, one should also note that if the organization has a very well-suited software architecture, even radically changing the type of game can have only a minor impact on the resulting software architecture.

This is not to say that the creative team specifies the software architecture. As shown in both the questionnaire and the survey, the creative team is allowed to participate in the design of software architecture, but usually only do this indirectly. There are of course different ways to indirectly affect the software architecture, one of which is through the discussion of implementing tools for the creative team. What features of the game engine should be exposed to allow this tool to do its job? How should this tool interact with the rest of the software suite?

Usually these sorts of changes are thoroughly discussed between the creative team and the technical team. This leads to a result which is the optimal compromise between creative freedom and technical limitations. During the development of these tools and interactions, the creative team provides feedback to the technical team, which again influences the software architecture.

The creative team is also allowed to request new features during development, which allows a dynamic evolution of the features. These features become, with continuous development and feedback, better and better suited to produce the game the organization is developing. However, there should of course be some restraint when implementing new features, as there is always a cost-benefit trade-off looming.

When large changes are to be implemented in the game, the process becomes more formal. From both the questionnaire and the survey one can see that the creative team can request changes, but that it cannot necessarily demand these changes. Often the creative team is given much weight in the decision whether or not to implement a feature, as the creative team should be able to judge if this is a make-or-break feature. Most often, however, the decision is a result of a dialogue between the creative team, the technical team, and management. In this discussion technical limitations, creative goals, end-user experience, cost, etc. are discussed, and can result in a specification of a prototype of the feature. If this works well, the prototype can become part of the game engine, as a direct result of the creative team's suggestion.

11.3 Research Question 2

The second RQ is:

> Are there any particular architectural approaches that facilitate the creative processes in game development?

The research performed in this thesis identifies several ways in which architectural approaches, both in terms of design and implementation, facilitate the creative processes in game development.

One interesting result of the research is that the respondents only reluctantly admitted that performance was the main goal of their software architecture. A game engine is a result of a constant trade-off between different qualities. Examples of such qualities include "the game *must* run smoothly", "the creative team has to be able to import new assets without the help of the technical team", "and there must be a scripting system for in-game events".

However, the most important result in this RQ is that a typical game engine is separated into two layers; core modules and gameplay modules. Such a layered approach was identified in Guldbrandsen and Storstein [13], and has been confirmed in the research presented in this thesis.

The core modules, which often include rendering and physics, are relatively stable, and can be reused across several games. These modules are also some of the most performance intensive modules, and an increased optimization, with the cost this entails, is rightly justified. This layer is intended to be developed and used by the technical team.

The gameplay layer on the other hand, is directed at the creative team. This layer should be used to present and develop the specifics of the game being produced. It will often present tools which the creative team can use to do their job, e.g., a scripting system for adding events to levels and actors.

There are also other approaches which are being used to support the creative processes. The questionnaire shows that adding new elements, both gameplay elements and more dynamic elements, such as Non-Player Characters (NPCs) or in-game items, is a simple process. When the creative team works with new ideas they can import new assets and script new events without interacting with the technical team at all.

When structured into rapid prototyping, this saves a lot of time for both the creative team and the technical team, since the creative team can work independently of any other members of the organization. Another great advantage is that the ideas can be tried out as early as possible, enabling the creative team to at once discard ideas which do not work. This increases the efficiency and is a direct result of how the software architecture is implemented.

Another trend in modern game engines is an increased use of middleware (more on this in Section 11.6). This is mainly done to improve the quality

of features in a cost-efficient manner. A direct benefit of this is that the technical team then can prioritize their time to support the creative team. Although this is a purely indirect effect, it is nonetheless an architectural approach which supports the creative team.

Lastly, the research shows that game engines and their supporting tools will continue to promote ease of use, allowing the creative team to perform more and more of the work related to creating a new game without interacting with the technical team. This is not to say that the technical team will be superfluous, but that the relationship between the creative team, the technical team, and the software will change.

11.4 Research Question 3

The third RQ is:

> Are there any particular development methods or processes which help the creative team do their job?

The research has uncovered three separate processes which all support the creative team and helps them do a better job.

The first is continuously changing requirements, often manifested in an iterative development process. This has been discussed on Nordmark [18] to some extent, and is further confirmed by the results of the questionnaire.

By allowing the creative team to request new features and tools along the way, it becomes unnecessary to specify all the required features before development has begun. This saves the creative team a lot of time in the beginning of a project, and also minimizes the number of "unnecessary", or superfluous, features which are implemented, which in turn saves the technical team from spending time developing these features.

The creative team is also included in the feedback loop during development, and this allows the technical team to produce tools and features which are tailored to the creative team's needs.

The second process is rapid prototyping. This has already been discussed in Section 11.3, so only a short summary in light of the third RQ will be presented here.

Rapid prototyping allows the creative team to try out ideas in the game at a very early stage. This allows the creative team to only design or model the bare essentials, import this to the game, and possibly script certain events in the game engine. By enabling the creative team to do this themselves, the team can discard bad or unsuitable ideas earlier, and refine good and suitable ideas quicker. This drives development of the game, and maximizes the time spent on developing content that actually ends up being used in the game.

The third process is how the companies handle large changes. If they are accepted, the organizations go through a process consisting of four steps. In chronological order these four steps are:

1. Firstly, the creative team and the technical team discuss the idea as presented by the creative team. This idea is then subjected to technical limitations, and both the technical team and the creative team have to accept compromises. The result from this step should be a specification of the feature which both the creative team and the technical team are happy with.

2. Then the technical team should transform this specification to a prototype implementation of the feature.

3. After the prototype is ready, the creative team tries out the functionality and assesses whether or not it is suitable. If there are features which are missing, the technical team add these. The result of this step should be a complete prototype the creative team is happy to work with.

4. In the fourth and last step, the technical team implements this prototype in production quality code, and it is permanently added to the software suite.

Each of the three processes mentioned above help the creative team do their job, either by allowing them to be as efficient as possible or by improving the workflow in general.

11.5 Research Question 4

The fourth RQ is:

> Do the organizations prioritize the needs and wants of the creative team?

The research shows that there are several different ways in which the creative team will be prioritized, e.g., by having features made available which are primarily intended for the creative team's tasks. These features range from including a scripting system which the creative team can use without involvement of the technical team, to allowing the automatic import of new assets directly from the tools. This again enables the creative team to perform rapid prototyping on their own, improving the workflow.

In addition to this, the creative team can request new features or tools during development of the game. If the creative team needs a particular feature or module, this can be requested. The research shows that game

development organizations will try to implement these if the cost-benefit trade-off is beneficial.

When requests for large changes are made, there will be a more formal decision process. Due to economical concerns, management usually has the final say in such decisions, but both the creative team and the technical team will be included in the discussions. During this process the idea presented by the creative team will be subjected to both technical limitations and financial concerns, and the creative team will contribute to the final decision; should this change be implemented or not?

If new features are implemented based on a request from the creative team, the creative team will be included in the feedback loop, allowing the technical team to tailor the feature to the actual need of the creative team.

When all this is seen together, one can see that the organizations will not cater to the creative team's every need. However, the creative team is able to request new features and tools, and the organization will include the team when deciding if, and how, these features will be implemented. Thus, the creative team will be prioritized, but will not receive any elevated status from which they could have demanded whatever they wanted.

11.6 Research Question 5

The fifth and final RQ in this thesis is:

> How has game development evolved over the years as an engineering discipline?

Traditionally, the game industry is considered to be "different" from other parts of software engineering. In the early days, the games were written in assembly, and did not utilize the operating system. Over the years, the industry has not gotten completely rid of this reputation, and is often still considered an immature field in software engineering.

However, the research presented in this thesis shows that game development has evolved towards many of todays high-held principles of software engineering (e.g., reuse).

The research has shown that the use of middleware and other third part software (such as complete game engines), have increased over the years. This implies a higher focus on the achieved quality for a certain amount of money. Buying or leasing a physics module will most likely produce better results than an in-house module at the same cost. Furthermore, when a company buys a third party engine, this engine will most likely include several other third party modules. This really shows that reuse of entire components has become an integral part of game development, putting it at least head-to-head with the rest of the software industry in this area.

Furthermore, there is a separation between the engineering aspects of game development and the creative aspects of game development. The engineering aspects of game development have become simpler over the years. The reasons for this is not explicitly identified in the research, but it is likely to assume that it arises partly from the increased reuse and the two-tier software architecture (core versus gameplay modules), as well as a of other factors.

On the creative aspects, however, the difficulty and complexity has increased. There is a host of different game companies, large and small, competing in the game market. As each of these companies always try to create a completely new experience, the projects have become more complex and difficult to complete. This is an area of game development in which much interesting research can be performed.

In general, however, the game industry is becoming a more mature software engineering discipline. The research discovered four characteristics which will become more and more important for game engines. These four characteristics are as follows:

Simplicity of use: Future game engines will continue to improve in usability. They will become easier and easier to use, allowing many game development organizations to create and publish a complete game without necessarily having any significant game engine development skills.

Completeness of the software suite: Game engines today already provide several integrated features. Future game engines which hope to compete in the market, will need to provide most, if not all, the features and tools which a company needs to develop a game. In addition, these tools and features must interoperate seamlessly, allowing the organization to work as efficiently as possible.

High-Quality Features: Today game engines are constantly affected by feature creep. Although new features are nice, it will become more important that the features which are delivered are both robust and that they perform as specified.

Multi-Platform: The game engines and their supporting tools will need to support deployment to several platforms without requiring the organization to spend much time adapting the game to these different platforms.

Game engines are expected to continue to evolve towards these four characteristics in the future.

Chapter 12

Future Studies

In this concluding chapter of the thesis, a few suggestions for future studies will be presented.

12.1 High-Level Third Party Game Engines

This thesis has uncovered that there is a trend toward more and more high-level use of third party game engine. Buying these high-level game engines allow organizations to put more effort into creating the content for the game (i.e., the levels, models, story, etc.) instead of having to focus on software development difficulties.

To what extent this is already done, or how the game industry is feeling about this change, has not been studied in this thesis, but should prove interesting areas of study.

12.2 Cost-Benefit Trade-Off

As mentioned several times during the thesis, game companies have to consider adding new features, tools, or other items based on a cost-benefit trade-off. This process is discussed in part in the analysis of S8 and S9 on Page 86 and 87, as well as other places in the thesis.

It would be interesting to continue the study of this process, and how this it is used in large companies as well as in smaller ones.

12.3 Feature Availability in Game Engines

As shown in this thesis, third party game engines are becoming more and more prevalent. It would be interesting to chart which features are in use in todays game engines, as well as how this feature availability has affected the competitive market of game engines. E.g., are there any particular features

of certain game engines which have proved so useful that the rest of the industry has had to follow suit?

12.4 Reference Architectures

Lastly, it would be very interesting to look into if there are any identifiable reference architecture for different games. E.g., are there particular architectural approaches which suit puzzle-games, or perhaps FPS-games? How will these approaches vary if the game should be single or multi player?

Bibliography

[1] Victor Basili. The Experimental Paradigm in Software Engineering. In H. Rombach, Victor Basili, and Richard Selby, editors, *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, volume 706 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 1993.

[2] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. *Encyclopedia of Software Engineering*, volume 1, chapter The Goal Question Metric Approach, pages 528–532. John Wiley & Sons, 1994.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., 2 edition, 2003.

[4] Jonathan Blow. Game Development: Harder Than You Think. *Queue*, 1(10):28–37, February 2004.

[5] David Budgen and Pearl Brereton. Performing Systematic Literature Reviews in Software Engineering. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 1051–1052. ACM, 2006.

[6] David Callele, Eric Neufeld, and Kevin Schneider. Requirements Engineering and the Creative Process in the Video Game Industry. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 240 – 250, aug. – sept. 2005.

[7] Crytek GmbH. CryENGINE, 2012. URL `http://www.crytek.com/cryengine`. Viewed on: 2012-04-30.

[8] Epic Games, Inc. Game Engine Technology by Unreal, 2012. URL `http://www.unrealengine.com/`. Viewed on: 2012-04-30.

[9] Epic Games, Inc. UnrealScript Object Oriented Programming Language, 2012. URL `http://www.unrealengine.com/features/unrealscript/`. Viewed on: 2012-04-30.

[10] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

[11] Firelight Technologies Pty. fmod: Interactive Audio Middleware, 2012. URL `http://www.fmod.org/`. Viewed on: 2012-04-30.

[12] Jason Gregory. *Game Engine Architecture*. A K Peters, 2009.

[13] Kjetil Guldbrandsen and Kjell Ivar Storstein. Apocalypse Engine: A Study of Software Architecture and Conventions in Modern Game Engines. Technical report, Norwegian University of Science and Technology, 2009.

[14] Havok.com Inc. Havok physics, 2012. URL `http://www.havok.com/`. Viewed on: 2012-05-01.

[15] Phillippe Kruchten. Architecture Blueprints – The "4+1" View Model of Software Architecture. In *Tutorial proceedings on TRI-Ada '91: Ada's role in global markets: solutions for a changing complex world*, pages 540–555. ACM, 1995.

[16] Audun Kvasbø. Postmortem analysis of video games. Technical report, Norwegian University of Science and Technology, 2006.

[17] Peter Naur and Brian Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*. 1969.

[18] Njål Nordmark. Software Architecture in Games. Technical report, Norwegian University of Science and Technology, 2011.

[19] Dewayne E. Perry and Alexander L. Wolf. Foundations for the Study of Software Architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52, October 1992.

[20] Andrew M. Phelps and David M. Parks. Fun and Games: Multi-Language Development. *Queue*, 1(10):46–56, February 2004.

[21] Andrew Rollings and David Morris. *Game Architecture and Design: A New Edition*. New Riders Games, 2003.

[22] SurveyMonkey.com, LLC. SurveyMonkey: Free online survey software & questionnaire tool, 2012. URL `http://www.surveymonkey.com/`. Viewed on: 2012-05-10.

[23] UBM Events Registration Dept. Game Developers Conference, 2012. URL `http://www.gdconf.com/`. Viewed on: 2012-04-25.

[24] UBM TechWeb. Game Developer Magazine, 2012. URL `http://www.gdmag.com/`. Viewed on: 2012-05-08.

[25] Unity Technologies. Unity – Game Engine, 2012. URL `http://unity3d.com/`. Viewed on: 2012-04-30.

[26] Paula Vicente and Elizabeth Reis. Using Questionnaire Design to Fight Nonresponse Bias in Web Surveys. *Social Science Computer Review*, 28(2):251–267, 2010.

[27] Jeff Ward. What is a Game Engine?, April 2008. URL `http://gamecareerguide.com/features/529/what_is_a_game_.php`. Viewed on: 2012-04-30.

[28] Stephen White. Postmortem: Naughty Dog's Jak & Daxter: The Precursor Legacy. *Game Developer Magazine*, pages 48 – 58, April 2002.

[29] Wikipedia. History of video games. URL `http://en.wikipedia.org/wiki/History_of_video_games`. Viewed on: 2012-05-06.

[30] R. Wirfs-Brock and B. Wilkerson. Object-Oriented Design: A Responsibility-Driven Approach. *SIGPLAN Not.*, 24:71–75, September 1989.

[31] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

Appendix A

Questionnaire

In this appendix, the questionnaire as sent to the game developers will be presented.

Firstly, the paper questionnaire will be presented. After this, the e-mail sent to game developers will be presented, immediately followed by the web questionnaire.

Do note that the title of the thesis has changed after the questionnaire was designed. The original title, which can be seen below, was "Using Software Architecture to Support the Creative Process in Game Development", whilst the new title is "Software Architecture and the Creative Process in Game Development."

A.1 Paper Questionnaire

Please see the next two pages for the paper questionnaire.

Software Architecture and the Creative Processes in Games

Introduction

This questionnaire is related to a study at the Norwegian University of Science and Technology titled ”Using Software Architecture to Facilitate the Creative Processes in Game Development”.

In this survey the questions will relate to opinions and attitudes regarding both software architecture and how the creative team affects and is affected by it.

In this questionnaire a distinction is implied between ”the technical team” and ”the creative team”. The definitions of these are given below. Please note that these teams are thought of as roles, not necessarily that every employee of your company has to be in one or the other.

The technical team has the role of technical implementer (e.g., software developer)	**The creative team** has the role of developing the story and content of the game (e.g., artist)

Two other definitions:

Game engine is the software driving the content regardless of whether it suits the formal definitions of a game engine.

Management is people, or personnel, which is responsible for the schedule and shipping of a game.

Most of the questions are phrased as statements. They will be presented with a scale (described below) as well as a field for optional comments (indicated by a **C:**). Where questions concern the creative team’s effect on the software architecture, this can be indirect effects like requesting a new particle system.

Scale:	Fully agree (FA), Partially agree (PA), Neutral (N), Partially disagree (PD), Fully disagree (FD), Not applicable (NA)

You and Your Company

Your Name:	
Your e-mail address:	
Your position in the company:	
Which company do you represent:	

Do you wish for your answers to be anonymized before publishing?	Yes ☐	No ☐

The number of employees in the company:

1 – 5 ☐	5 – 10 ☐	10 – 20 ☐	20 – 50 ☐	50 – 100 ☐	100 – 500 ☐	500+ ☐

Which genres do you develop games in:	
Which platforms do you develop games for:	

Design of Software Architecture

Design of the software architecture is an important part of our game development process.

FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:

The main goal of our software architecture is performance.

FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:

Our game concept heavily influences the software architecture.

FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:

The creative team is included in the design of the software architecture.

FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:

Our existing software suite provides features aimed at helping the creative team do their job.

FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:

Our existing software architecture dictates the future game concepts we can develop.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Changes to the Software Architecture during Development						
The creative team has to adopt their ideas to the existing game engine.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
During development, the creative team can demand changes to the software architecture.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Who decides if change-requests from the creative team are implemented?						
The technical team ☐			Management ☐			The creative team ☐
The technical team implements all features requested by the creative team.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
It is simple to add new gameplay elements after the core of our game engine has been completed.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
During development, the creative team has to use the tools and features already available.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Supporting the creative processes						
Our game engine supports dynamic loading of new content.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Our game engine has a scripting system the creative team can use to try out and implement new ideas.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
The creative team is included in our development feedback loop (e.g., scrum meetings).						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Changes over Time						
Today our company uses more 3rd party modules than 3 years ago.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Examples of 3rd party software we use:						
It is easier to develop games today than it was 5 years ago.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Middleware is more important to our company today than 3 years ago.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Game development is more like ordinary software development today than 5 years ago.						
FA ☐	PA ☐	N ☐	PD ☐	FD ☐	NA ☐	C:
Closing Remarks						
Are you willing answer some more in-depth follow up questions later?				Yes ☐		No ☐
Would you like to receive the research when it is completed?				Yes ☐		No ☐
Any other information or comments:						

A.2 E-Mail sent to Game Developers

The e-mail sent to game developers is given in full below:

```
Dear Sir/Madam

I am a computer science masters student at the Norwegian
University of Science and Technology (NTNU), conducting
research on software architecture and games. My goal is to
learn how the creative processes of game development are
supported by the software architecture through asking game
developers about their practices.

At the end of this e-mail you will find a link to an online
survey which will take between 5 and 10 minutes to complete.

Most of the questions are phrased as statements. They will be
presented with a scale as well as a field for optional
comments. Where questions concern the creative team's effect
on the software architecture, this can be indirect effects
like requesting a new particle system which leads to a change
in the software architecture.

A few important definitions:

* Game engine is the software driving the content regardless
  of whether it suits the formal definitions of a game engine
  or not.
* Management has the role which is responsible for the
  schedule, economy, and shipping of a game.
* The technical team has the role of developing the software
* The creative team has the role of developing the story, art,
  and content of the game (e.g., artist)

Please note that management and technical and creative team
are thought of as roles, not necessarily that every employee
of your company has to be in one or the other.

Survey:
  https://www.surveymonkey.com/s/CreativeSoftwareArchitecture

--
Njål Nordmark
```

A.3 Web Questionnaire

Here screen shots of the entire web questionnaire will be presented in order.

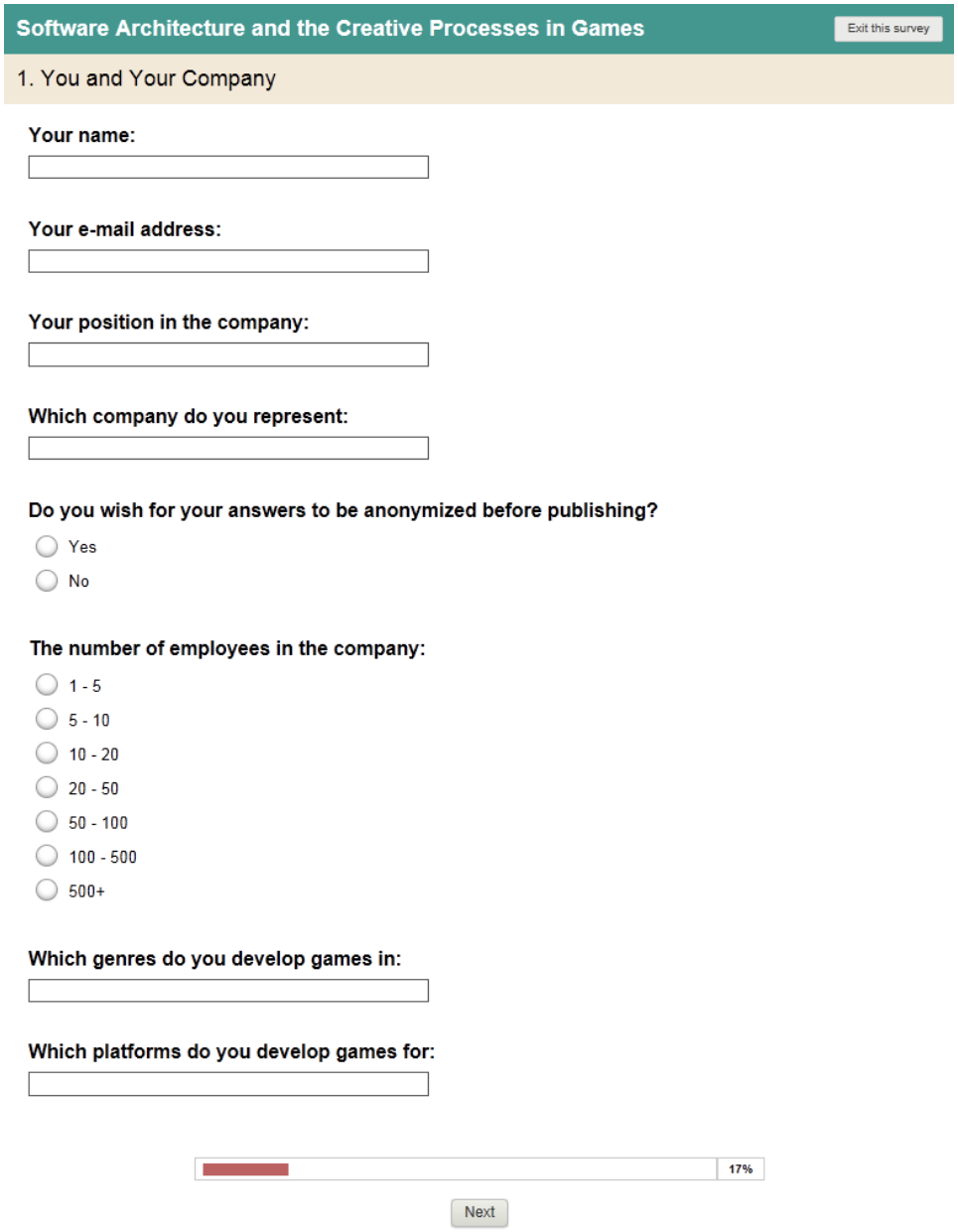

Figure A.1: First section of the web questionnaire: "You and Your Company"

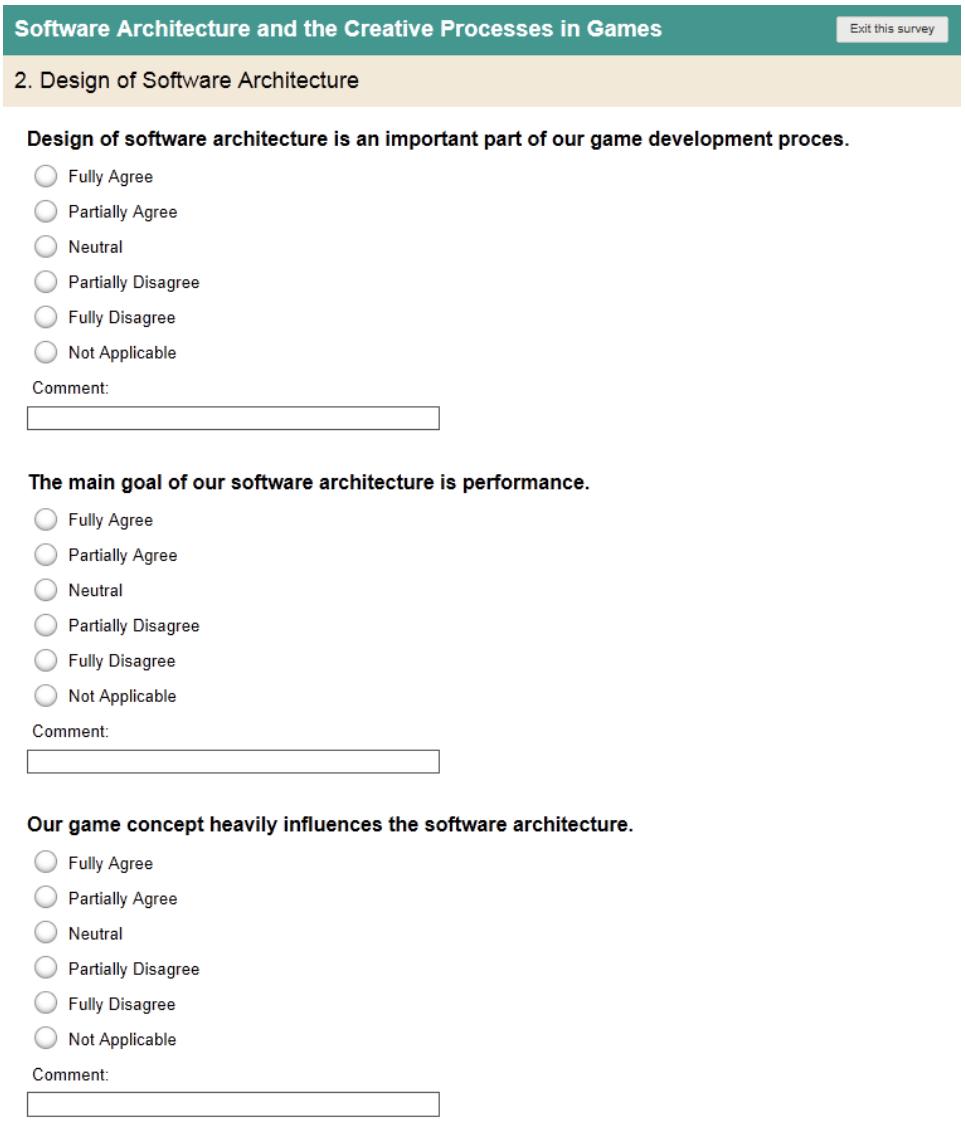

Figure A.2: First part of the second section of the web questionnaire: "Design of Software Architecture"

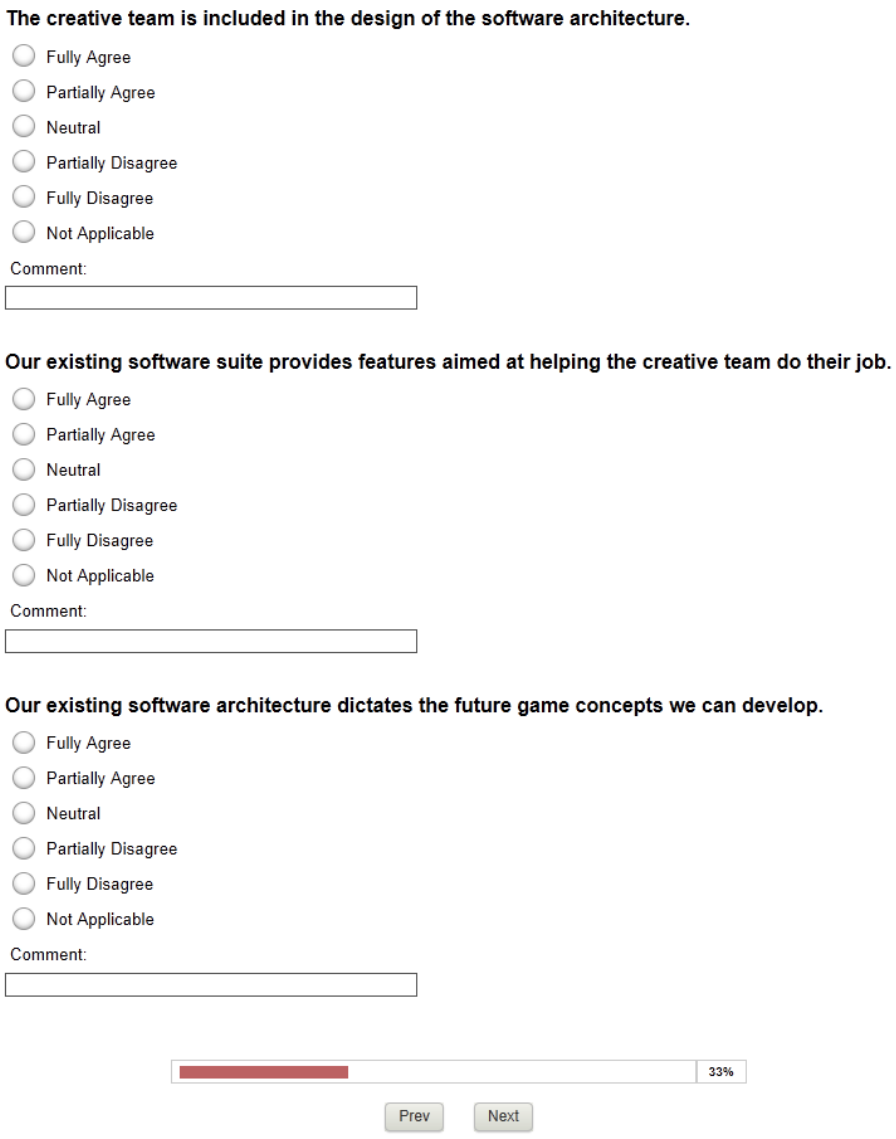

Figure A.3: Second part of the second section of the web questionnaire: "Design of Software Architecture"

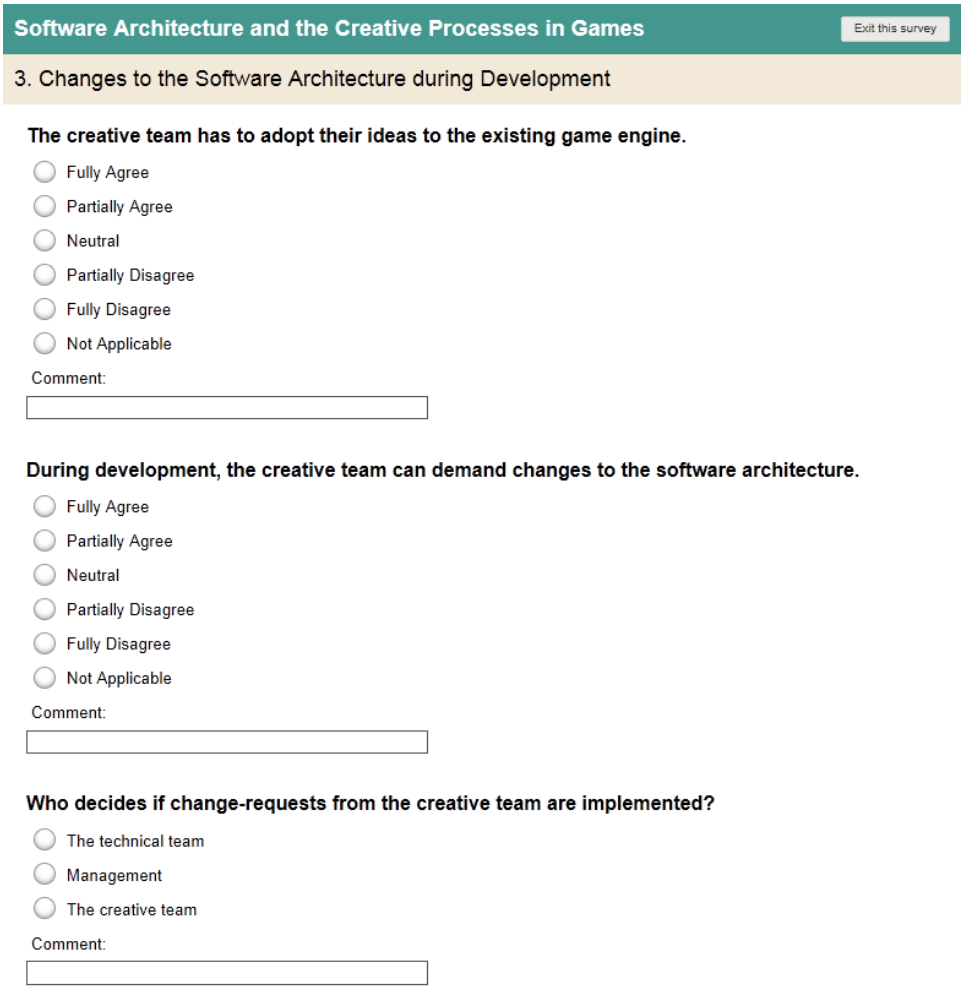

Figure A.4: First part of the third section of the web questionnaire: "Changes to the Software Architecture during Development"

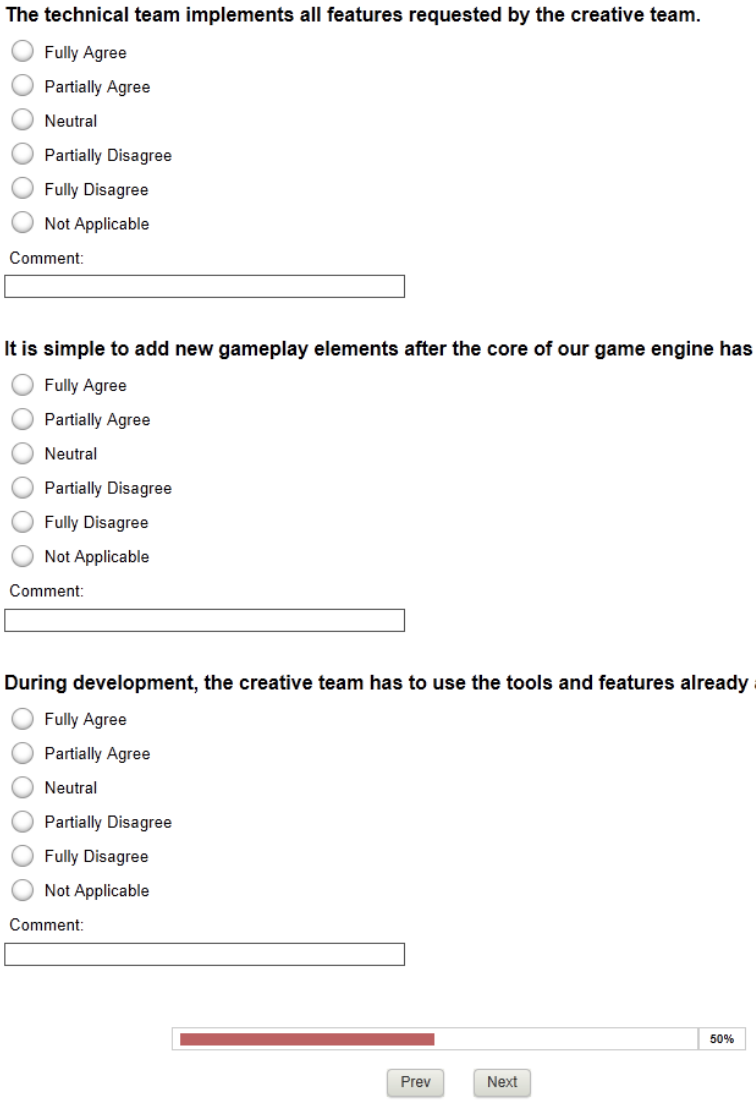

Figure A.5: Second part of the third section of the web questionnaire: "Changes to the Software Architecture during Development"

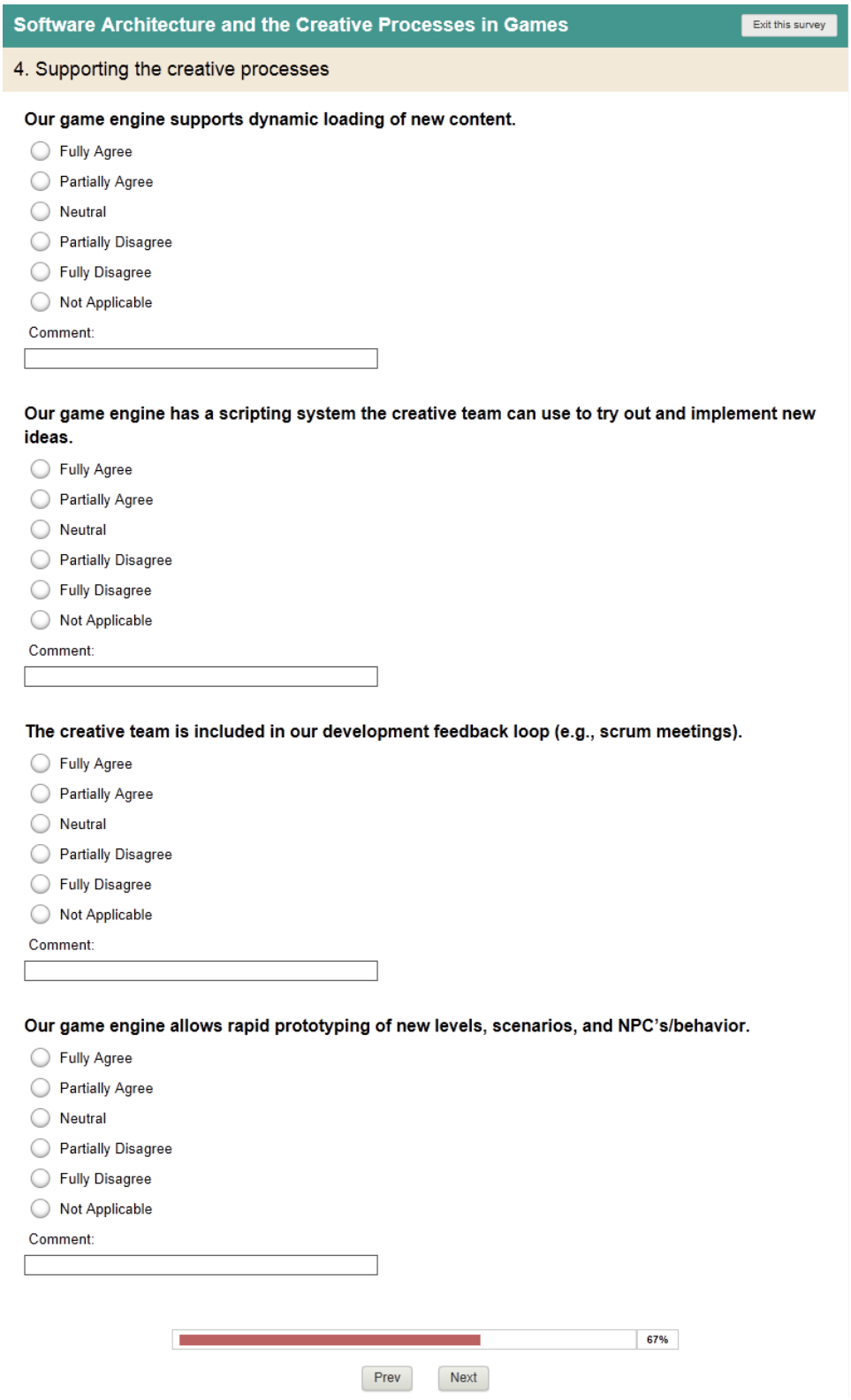

Figure A.6: The fourth section of the web questionnaire: "Supporting the Creative Processes"

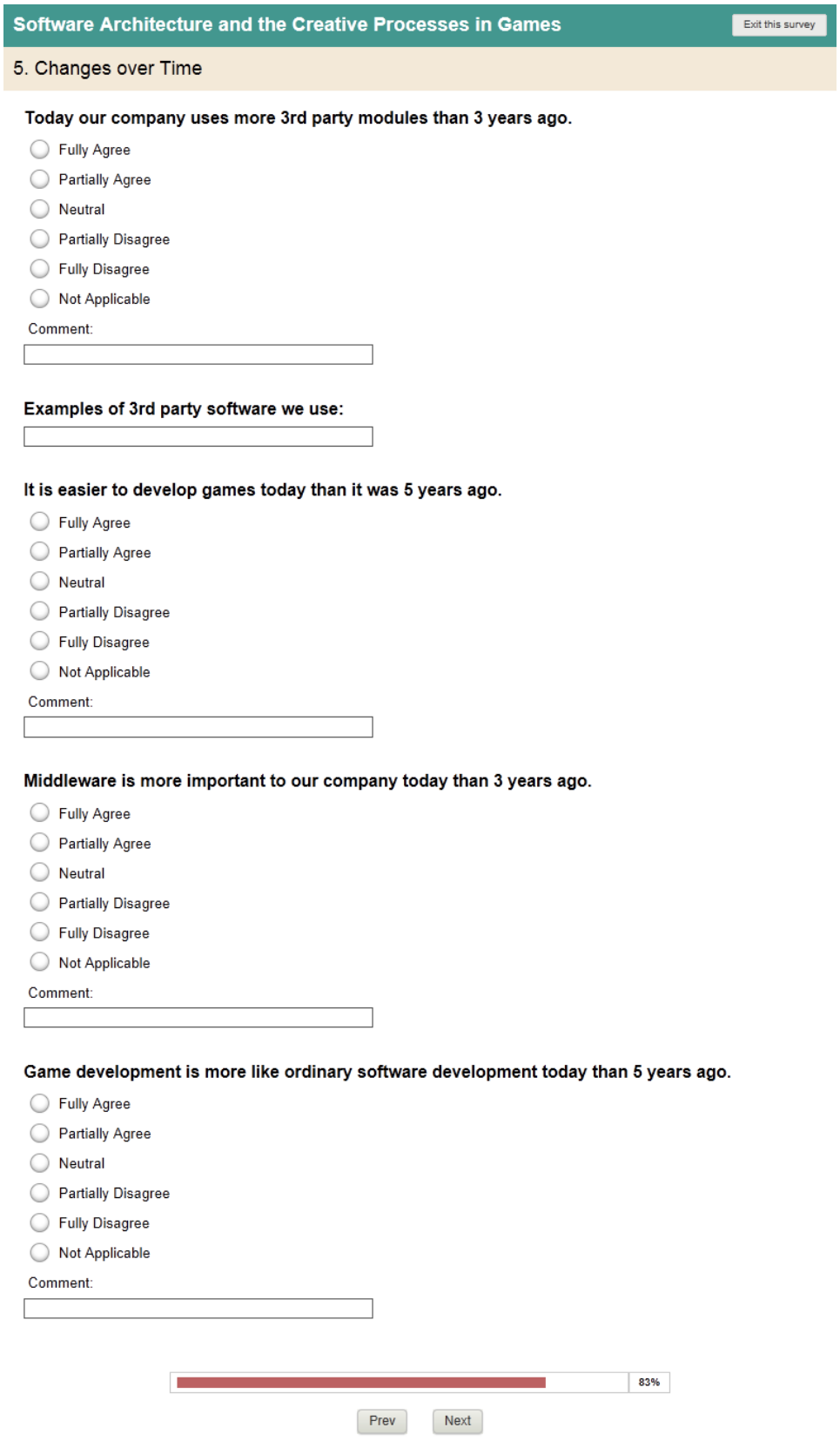

Figure A.7: The fifth section of the web questionnaire: "Changes over Time"

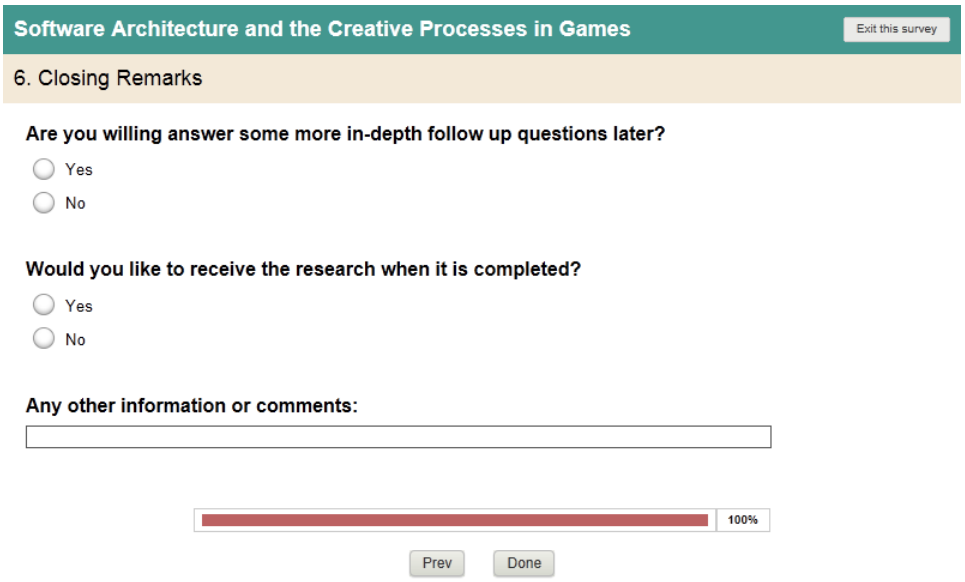

Figure A.8: The sixth and final section of the web questionnaire: "Closing Remarks"

Appendix B

Questionnaire Results

In this chapter the results of the questionnaire will be presented. To preserve the respondents' anonymity, the companies will be named "Company A", "Company B", etc.

B.1 Company A's Questionnaire Response

Table B.1: Company A's Questionnaire Results

Company A	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Q2: The main goal of our software architecture is performance.	Partially Agree
Comment:	Performance plus functionality.
Q3: Our game concept heavily influences the software architecture.	Fully Agree
Q4: The creative team is included in the design of the software architecture.	Fully Agree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Partially Agree
Comment:	Our third party tools do not do this, but we've developed in-house extensions that do.

Company A	
Q6: Our existing software architecture dictates the future game concepts we can develop.	Partially Disagree
Comment:	It makes it a bit more expensive to go to certain genres, but that's it.

Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Partially Agree
Comment:	Technical realities are always something the creative side has to work around.
Q8: During development, the creative team can demand changes to the software architecture.	Fully Agree
Q9: Who decides if change-requests from the creative team are implemented?	Management
Q10: The technical team implements all features requested by the creative team.	Neutral
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Fully Agree
Q12: During development, the creative team has to use the tools and features already available.	Fully Disagree

Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Neutral
Examples of 3rd party software we use:	Unity 3D and various modules for it.

Company A	
Q18: It is easier to develop games today than it was 5 years ago.	Partially Agree
Q19: Middleware is more important to our company today than 3 years ago.	Fully Disagree
Q20: Game development is more like ordinary software development today than 5 years ago.	Partially Agree

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	No
Any other information or comments:	

B.2 Company B's Questionnaire Response

Table B.2: Company B's Questionnaire Results

Company B	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Comment:	Oversights in the game software architecture may lead to serious dead ends leading to rewrite of entire systems
Q2: The main goal of our software architecture is performance.	Partially Agree
Comment:	Main goals are: 1. performances 1.5. memory consumption 2. actual purpose of the software. Real time softwares as games *must* perform according to the platform requirements in order to see the light of the day regardless of the content ;)
Q3: Our game concept heavily influences the software architecture.	Partially Agree
Comment:	Entirely depends on the game concept requirements but in general: the more generic, within boundaries, the better.
Q4: The creative team is included in the design of the software architecture.	Partially Agree
Comment:	This is mostly true when working on the tools the creative team will be using. It rarely applies to in-game specific features.
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Neutral
Comment:	It may influence, but not dictate whenever possible

Company B	
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Partially Disagree
Comment:	Most of the time, the creative team is not fully aware of the game engine limitations so it is not their job to make it work by locking the creativity to things known to have been done with the engine before, the people who implements just need to make the ideas work one way or another :)
Q8: During development, the creative team can demand changes to the software architecture.	Neutral
Comment:	Depends how far in development and how big of a changes, the odds of re-factoring an entire system late in production are close to nil, but the development team keeps an open mind at all times.
Q9: Who decides if change-requests from the creative team are implemented?	Management
Comment:	Ultimately, the management can overrule everybody, but I would like to check the 3 options here, the creative team judges how important the change is, the technical team decides if it is realistic and the management makes sure it can be afforded. So mostly, it is a team decision.
Q10: The technical team implements all features requested by the creative team.	Partially Agree
Comment:	It can happen the creative team contributes on technical aspects during prototyping phase. Production quality code is however left to the technical people.

Company B	
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Fully Agree
Comment:	It is simple during prototyping phase, technology-wise. However from a game concept point of view, it is highly disrecommended and the fact it is simple does not motivate the team to stack up features because the existing one are just not convincing enough :)
Q12: During development, the creative team has to use the tools and features already available.	Partially Agree
Comment:	The ones already available and the ones they request along the way.

Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Partially Agree
Comment:	At some extent, in editor mode yes, at runtime only a subset of it.
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Partially Agree
Comment:	At some extent, in editor mode yes, at runtime only a subset of it.
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Neutral
Comment:	Depends on the phase of the project.
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Partially Agree
Comment:	While most of the systems are designed with simplicity and fast iteration time in mind, certain things still requires time consuming tweaking tasks

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Agree
Comment:	It is about time... ;)

Company B		
Examples of 3rd party software we use:	Unity (which includes Beast, Umbra, FMOD, Substance, NVidia PhysX) and some free to use public API	
Q18: It is easier to develop games today than it was 5 years ago.		Partially Disagree
Comment:	The challenges have changed and the quality bar has risen, it is more accessible to people less interested in nerdy things nowadays (engines like Unity reduced/removed the low-level aspect of the development), but developing a great game is still as challenging as before, the problems to solve just have evolved.	
Q19: Middleware is more important to our company today than 3 years ago.		Partially Agree
Q20: Game development is more like ordinary software development today than 5 years ago.		Partially Disagree
Comment:	Game development requires a more eccentric creative problem solving than development in most of other industries and this will probably remain true forever ;)	

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

B.3 Company C's Questionnaire Response

Table B.3: Company C's Questionnaire Results

Company C	
The number of employees in the company:	1 – 5
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Q2: The main goal of our software architecture is performance.	Fully Disagree
Q3: Our game concept heavily influences the software architecture.	Fully Agree
Q4: The creative team is included in the design of the software architecture.	Fully Agree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Partially Disagree
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Fully Disagree
Q8: During development, the creative team can demand changes to the software architecture.	Fully Agree
Q9: Who decides if change-requests from the creative team are implemented?	The creative team
Q10: The technical team implements all features requested by the creative team.	Fully Agree
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Partially Agree
Q12: During development, the creative team has to use the tools and features already available.	Neutral

Company C	
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Partially Agree
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Partially Agree
Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Not Applicable
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Not Applicable
Q20: Game development is more like ordinary software development today than 5 years ago.	Partially Agree
Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

B.4 Company D's Questionnaire Response

Table B.4: Company D's Questionnaire Results

Company D	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Q2: The main goal of our software architecture is performance.	Fully Disagree
Q3: Our game concept heavily influences the software architecture.	Fully Agree
Q4: The creative team is included in the design of the software architecture.	Partially Agree
Comment:	Only because I am a programmer and also the lead designer. Other "creative" people don't know enough to be productively included.
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Fully Disagree
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Fully Disagree
Comment:	That is not the way we do it here. The game design comes first, then we build what is necessary to make it happen.
Q8: During development, the creative team can demand changes to the software architecture.	Fully Agree
Comment:	But again, only because the head of the "creative team" is president of the company and also wrote the original version of the game engine. If someone who doesn't know how to program were to come to me and demand changes to the software architecture, I would probably not listen very seriously.

Company D	
Q9: Who decides if change-requests from the creative team are implemented?	Management
Comment:	Actually it is all of the above, but the question would not let me put that as an answer.
Q10: The technical team implements all features requested by the creative team.	Not Applicable
Comment:	Things just aren't segmented this way in our situation.
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Fully Agree
Q12: During development, the creative team has to use the tools and features already available.	Fully Disagree
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Partially Disagree
Comment:	Our "scripting system" is typing in C++ code and recompiling the game.
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Not Applicable
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Not Applicable
Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Disagree
Examples of 3rd party software we use:	Bink, libvorbis
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Fully Disagree
Q20: Game development is more like ordinary software development today than 5 years ago.	Fully Disagree

Company D	
Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	A lot of these questions seem inappropriate to our situation but I tried to answer them anyway.

B.5 Company E's Questionnaire Response

Table B.5: Company E's Questionnaire Results

Company E	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Q2: The main goal of our software architecture is performance.	Partially Agree
Q3: Our game concept heavily influences the software architecture.	Partially Disagree
Q4: The creative team is included in the design of the software architecture.	Partially Agree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Partially Agree
Comment:	Use two software tiers, that aims at very different levels of artist integration: Visual Studio and Unity3D
Q6: Our existing software architecture dictates the future game concepts we can develop.	Fully Disagree
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Neutral
Comment:	Depending on structure. For assets handling, yes, but creatively, not so much. In latter case, the challenge is put to programmers to extend useage
Q8: During development, the creative team can demand changes to the software architecture.	Partially Agree
Q9: Who decides if change-requests from the creative team are implemented?	The technical team
Comment:	Sort of. The technical team advice what is possible, and as such has final word. If it is possible, the decision falls on management, as it is usually related to economic costs

Company E	
Q10: The technical team implements all features requested by the creative team.	Fully Agree
Comment:	Of course, if the requests are decided to be implemented in the first place
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Neutral
Comment:	This really depends a lot, and can only be answered on a case to case effect
Q12: During development, the creative team has to use the tools and features already available.	Partially Disagree
Comment:	New tools can be made. However, it is certanly best to keep within the suite offered
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Partially Agree
Comment:	With some constraints, content must be propperly prepped of course
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree
Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Agree
Examples of 3rd party software we use:	A lot of stuff around Unity3D and the community there
Q18: It is easier to develop games today than it was 5 years ago.	Partially Agree
Comment:	Technically and graphically, yes. Conceptually, no.
Q19: Middleware is more important to our company today than 3 years ago.	Partially Agree

Company E	
Q20: Game development is more like ordinary software development today than 5 years ago.	Fully Disagree
Comment:	Nope. It was software development then, and still is now

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

B.6 Company F's Questionnaire Response

Table B.6: Company F's Questionnaire Results

Company F	
The number of employees in the company:	1 – 5
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Partially Disagree
Q2: The main goal of our software architecture is performance.	Neutral
Q3: Our game concept heavily influences the software architecture.	Neutral
Q4: The creative team is included in the design of the software architecture.	Neutral
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Neutral
Q6: Our existing software architecture dictates the future game concepts we can develop.	Neutral
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Neutral
Q8: During development, the creative team can demand changes to the software architecture.	Neutral
Q9: Who decides if change-requests from the creative team are implemented?	The creative team
Q10: The technical team implements all features requested by the creative team.	Fully Agree
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Fully Agree
Q12: During development, the creative team has to use the tools and features already available.	Fully Agree
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree

Company F	
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree
Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Agree
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Fully Agree
Q20: Game development is more like ordinary software development today than 5 years ago.	Neutral
Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	No
Any other information or comments:	

B.7 Company G's Questionnaire Response

Table B.7: Company G's Questionnaire Results

Company G	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Not Applicable
Q2: The main goal of our software architecture is performance.	Not Applicable
Q3: Our game concept heavily influences the software architecture.	Not Applicable
Q4: The creative team is included in the design of the software architecture.	Not Applicable
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Neutral
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Fully Agree
Q8: During development, the creative team can demand changes to the software architecture.	Partially Agree
Q9: Who decides if change-requests from the creative team are implemented?	The creative team
Q10: The technical team implements all features requested by the creative team.	Fully Agree
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Not Applicable
Q12: During development, the creative team has to use the tools and features already available.	Partially Agree
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree

Company G	
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Agree
Examples of 3rd party software we use:	UE3, Speedtree, Scaleform
Q18: It is easier to develop games today than it was 5 years ago.	Fully Disagree
Q19: Middleware is more important to our company today than 3 years ago.	Fully Agree
Q20: Game development is more like ordinary software development today than 5 years ago.	Fully Disagree

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	we do not research and produce our own engine but licence middleware and engine, we spend our coding time on special features and gameplay.

B.8 Company H's Questionnaire Response

Table B.8: Company H's Questionnaire Results

Company H	
The number of employees in the company:	500+
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Q2: The main goal of our software architecture is performance.	Partially Disagree
Comment:	also future change, ability to be datadriven, optimised deployment processes, ease ot automation/scriptability, testability
Q3: Our game concept heavily influences the software architecture.	Partially Agree
Q4: The creative team is included in the design of the software architecture.	Neutral
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Neutral
Comment:	We have engines that gives us a great benefit when building new games and we would preferr to continue on same engines, however it doesnt fully dictate the games we will make in future, this is primarily market driven
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Neutral
Q8: During development, the creative team can demand changes to the software architecture.	Fully Agree

Company H	
Q9: Who decides if change-requests from the creative team are implemented?	The creative team
Comment:	depends very much on the scale of change, we try as much as possible to keep this within and as a dialogue between the tech/creative teams, but if it means major change it goes to management. We also aim to be as much product/feature driven so primary owner is creative team.
Q10: The technical team implements all features requested by the creative team.	Partially Agree
Comment:	its very much a dialogue, we try not to have too formal split between tech and creative team when thinking about this, but priorotise what the user experience should be and when we can ship at target quality
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Partially Agree
Q12: During development, the creative team has to use the tools and features already available.	Neutral
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Partially Agree
Comment:	yes, but could be better and more flexible (as always...)
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree
Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Agree
Examples of 3rd party software we use:	software or modules?

Company H	
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Partially Agree
Q20: Game development is more like ordinary software development today than 5 years ago.	Neutral

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

B.9 Company I's Questionnaire Response

Table B.9: Company I's Questionnaire Results

Company I	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Fully Agree
Q2: The main goal of our software architecture is performance.	Partially Agree
Q3: Our game concept heavily influences the software architecture.	Fully Agree
Q4: The creative team is included in the design of the software architecture.	Fully Agree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Neutral
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Neutral
Q8: During development, the creative team can demand changes to the software architecture.	Partially Agree
Q9: Who decides if change-requests from the creative team are implemented?	
Q10: The technical team implements all features requested by the creative team.	Partially Disagree
Comment:	Some requested features are not tech. feasible
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Neutral
Comment:	Depends on the type of element - some may require significant underlying engine changes

Company I	
Q12: During development, the creative team has to use the tools and features already available.	Partially Disagree
Comment:	Our current engine (Unity) is easily extensible

Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Disagree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Fully Agree
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Neutral
Q19: Middleware is more important to our company today than 3 years ago.	Fully Agree
Q20: Game development is more like ordinary software development today than 5 years ago.	Fully Disagree
Comment:	I think the tools available today moves game dev further away from "ordinary software dev".)

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

B.10 Company J's Questionnaire Response

Table B.10: Company J's Questionnaire Results

Company J	
The number of employees in the company:	1 – 5
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Partially Agree
Q2: The main goal of our software architecture is performance.	Partially Agree
Q3: Our game concept heavily influences the software architecture.	Partially Agree
Q4: The creative team is included in the design of the software architecture.	Neutral
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Partially Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Neutral
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Neutral
Q8: During development, the creative team can demand changes to the software architecture.	Neutral
Q9: Who decides if change-requests from the creative team are implemented?	The creative team
Q10: The technical team implements all features requested by the creative team.	Partially Agree
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Fully Agree
Q12: During development, the creative team has to use the tools and features already available.	Partially Agree
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Neutral

Company J	
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Neutral
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Neutral

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Neutral
Q20: Game development is more like ordinary software development today than 5 years ago.	Partially Agree

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

B.11 Company K's Questionnaire Response

Table B.11: Company K's Questionnaire Results

Company K	
The number of employees in the company:	1 – 5
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Partially Agree
Q2: The main goal of our software architecture is performance.	Partially Agree
Q3: Our game concept heavily influences the software architecture.	Partially Disagree
Q4: The creative team is included in the design of the software architecture.	Fully Agree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Fully Disagree
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Neutral
Q8: During development, the creative team can demand changes to the software architecture.	Fully Agree
Q9: Who decides if change-requests from the creative team are implemented?	The technical team and the creative team
Q10: The technical team implements all features requested by the creative team.	Partially Agree
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Not Applicable
Q12: During development, the creative team has to use the tools and features already available.	Partially Agree
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree

Company K	
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Partially Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Not Applicable
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Partially Agree
Q20: Game development is more like ordinary software development today than 5 years ago.	Neutral

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	No
Any other information or comments:	

B.12 Company L's Questionnaire Response

Table B.12: Company L's Questionnaire Results

Company L	
The number of employees in the company:	5 – 10
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Neutral
Q2: The main goal of our software architecture is performance.	Partially Agree
Q3: Our game concept heavily influences the software architecture.	Partially Agree
Q4: The creative team is included in the design of the software architecture.	Partially Disagree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Fully Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Partially Agree
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Partially Agree
Q8: During development, the creative team can demand changes to the software architecture.	Neutral
Q9: Who decides if change-requests from the creative team are implemented?	The technical team, management, and the creative team
Q10: The technical team implements all features requested by the creative team.	Neutral
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Fully Agree
Q12: During development, the creative team has to use the tools and features already available.	Fully Disagree

Company L	
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Fully Agree
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Neutral
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree
Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Not Applicable
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Fully Agree
Q19: Middleware is more important to our company today than 3 years ago.	Not Applicable
Q20: Game development is more like ordinary software development today than 5 years ago.	Partially Agree
Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	No
Any other information or comments:	

B.13 Company M's Questionnaire Response

Table B.13: Company M's Questionnaire Results

Company M	
The number of employees in the company:	1 – 5
Design of Software Architecture	
Q1: Design of software architecture is an important part of our game development process.	Neutral
Q2: The main goal of our software architecture is performance.	Neutral
Q3: Our game concept heavily influences the software architecture.	Partially Agree
Q4: The creative team is included in the design of the software architecture.	Fully Agree
Q5: Our existing software suite provides features aimed at helping the creative team do their job.	Partially Agree
Q6: Our existing software architecture dictates the future game concepts we can develop.	Fully Agree
Changes to Software Architecture during Development	
Q7: The creative team has to adopt their ideas to the existing game engine.	Partially Agree
Q8: During development, the creative team can demand changes to the software architecture.	Partially Agree
Q9: Who decides if change-requests from the creative team are implemented?	Management
Q10: The technical team implements all features requested by the creative team.	Partially Agree
Q11: It is simple to add new gameplay elements after the core of our game engine has been completed.	Partially Agree
Q12: During development, the creative team has to use the tools and features already available.	Partially Agree
Supporting the Creative Processes	
Q13: Our game engine supports dynamic loading of new content.	Partially Agree

Company M	
Q14: Our game engine has a scripting system the creative team can use to try out and implement new ideas.	Fully Agree
Q15: The creative team is included in our development feedback loop (e.g., scrum meetings).	Fully Agree
Q16: Our game engine allows rapid prototyping of new levels, scenarios, and NPC's/behavior.	Fully Agree

Changes over Time	
Q17: Today our company uses more 3rd party modules than 3 years ago.	Neutral
Examples of 3rd party software we use:	
Q18: It is easier to develop games today than it was 5 years ago.	Partially Agree
Q19: Middleware is more important to our company today than 3 years ago.	Neutral
Q20: Game development is more like ordinary software development today than 5 years ago.	Partially Agree

Closing Remarks	
Are you willing answer some more in-depth follow up questions later?	Yes
Any other information or comments:	

Appendix C

Survey

In this section, the details of the survey are presented. Firstly the e-mail sent to game developers regarding the survey will be given. Secondly, screen shots of the web survey will be presented.

C.1 E-Mail sent to Game Developers

The e-mail sent to the developers is given in full below.

```
Dear Sir/Madam,

A while back you replied to a questionnaire regarding the
creative processes and software architecture in games. First
of all I would like to thank you for answering that
questionnaire. I really appreciate the time and effort you
have used.

Furthermore, at the end of the previous questionnaire you
indicated that you would be able to answer some follow-up
questions to said quesitonnaire. These follow-up questions are
avaialable at:
https://www.surveymonkey.com/s/FollowUpCreativeSoftwareArchitecture

Once again, thank you for replying to the questionnaire, and
thank you for your time.

--
Njål Nordmark
```

C.2 Web Survey

In this section screen shots of the web survey will be presented in order.

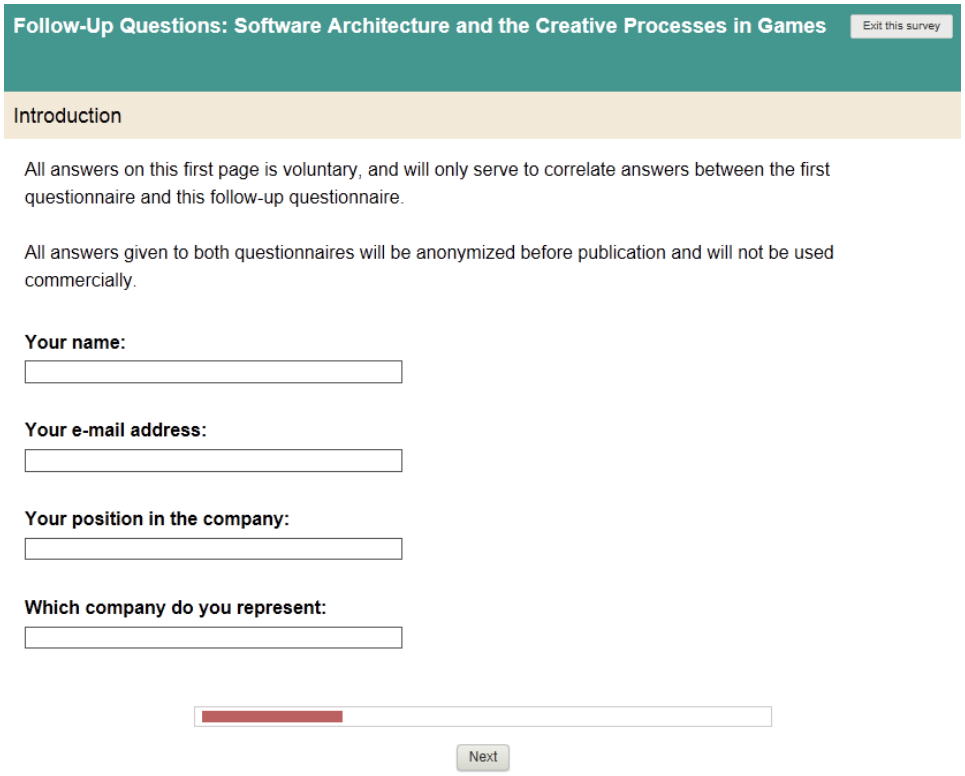

Figure C.1: First section of the web survey: “Introduction”

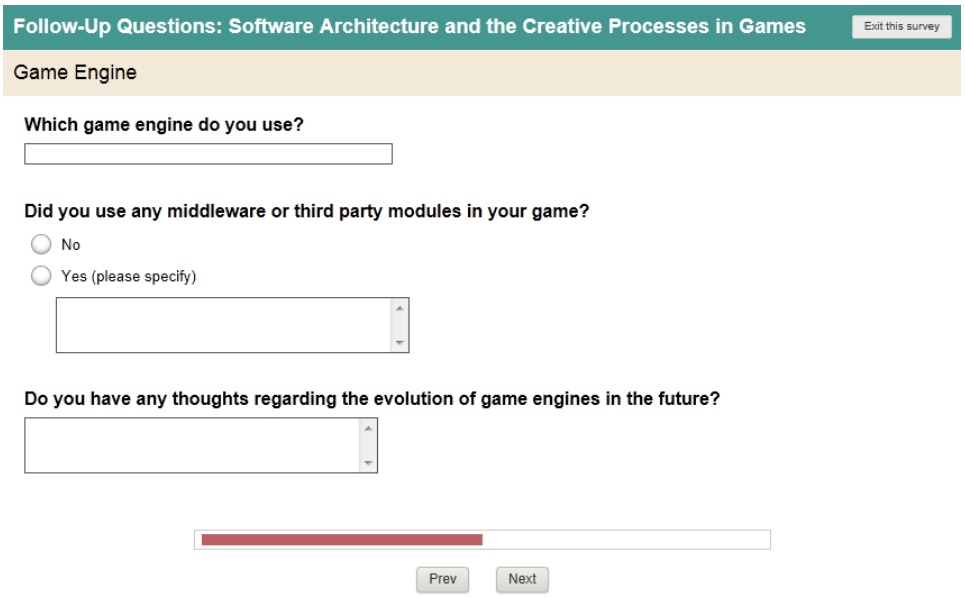

Figure C.2: First section of the web survey: “Game Engine”

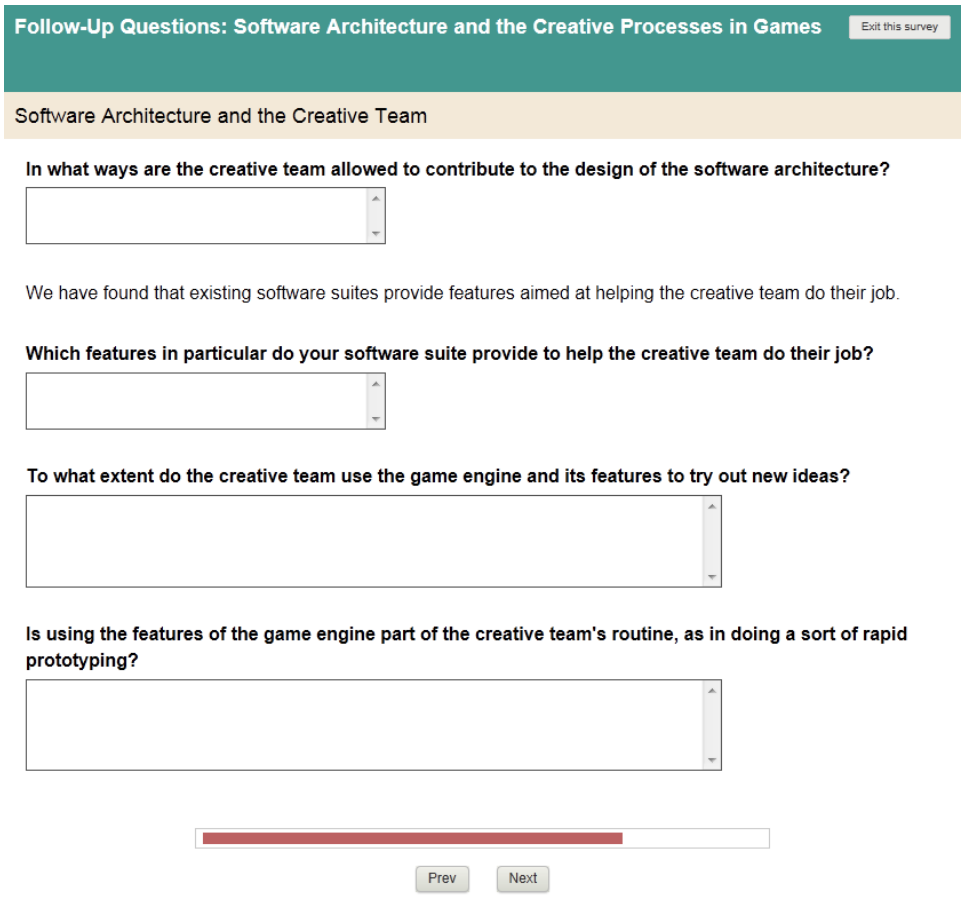

Figure C.3: First section of the web survey: "Software Architecture and the Creative Team"

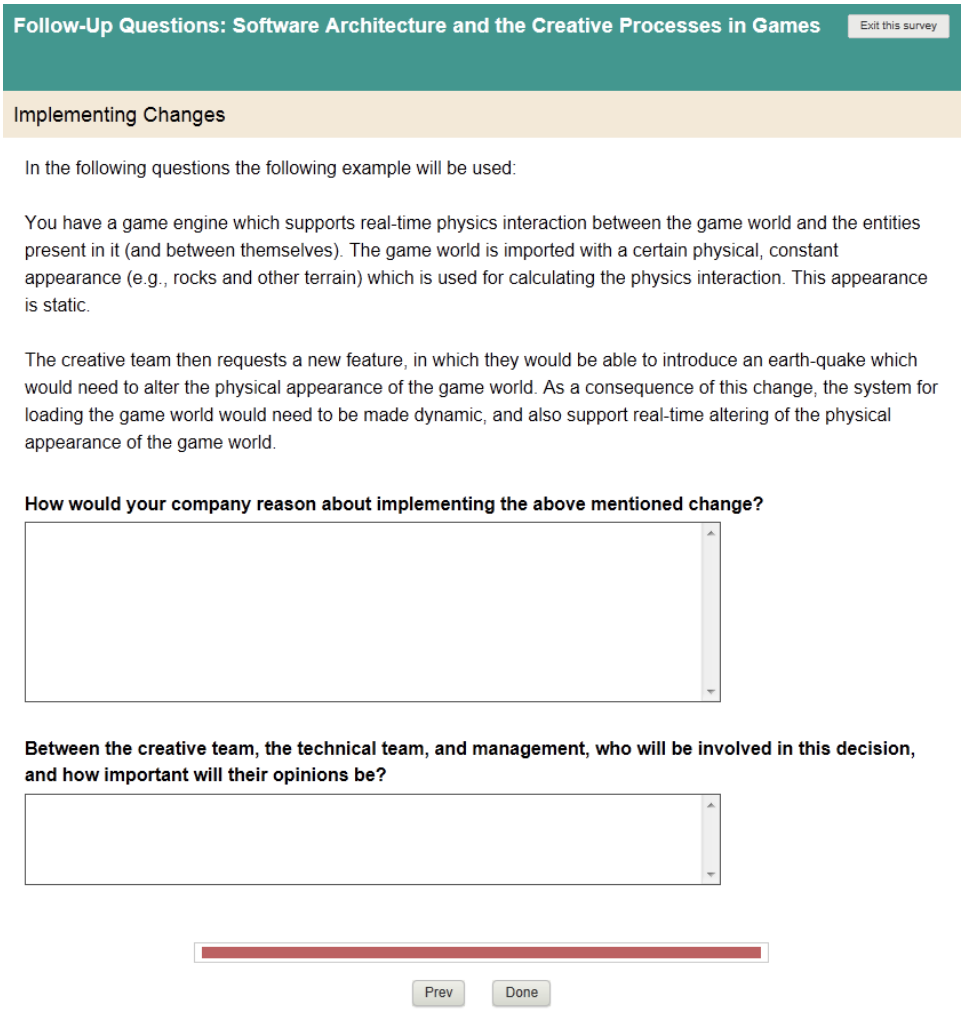

Figure C.4: First section of the web survey: “Implementing Changes”

Appendix D

Survey Results

In this appendix the results from the survey will be presented.

Where the responses have allowed connecting the replies from the questionnaire to the survey, the companies will receive the same name (e.g., Company B). The one respondent which did not provide the identifying information will be named Company Z.

D.1 Company B

Table D.1: Company B's Questionnaire Results

Company B	
Game Engine	
Which game engine do you use?	Unity 3D
Did you use any middleware or third party modules in your game?	
Yes (please specify): On the previous game, developed with the in-house engine, FMOD was the only third-party software licensed and used for audio playback. SDL was also used for the Mac OS X port but was a free open source library. On the next projects, done with Unity 3D, we obviously use Unity as middleware. Because of the way their engine is constructed, it is highly unlikely we will need any additional third party modules as FMOD, Beast lightmapping and Umbra occlusion culling solution are integrated to the engine and part of the engine license.	

Company B
Do you have any thoughts regarding the evolution of game engines in the future?
Very small game engines will most likely remain being used for very small budget, low profile projects the same way it has always been. With the big players, the competition is becoming tougher and tougher, Unreal Engine and Cryengine have been leading the market for a long time but their costs and licensing models are not best fitted for small companies which would be able to create AAA quality games even with a smaller budget than some bigger companies. Unity 3D has started to be seen as a dangerous competition as it is nowadays of about the same power than the big competitiors and after they turned their indie license into a free version, Unreal responded by bringing the UDK for indie developer, when Unity 3D came with robust mobile platform support and Flash, both Unreal and Cryengine came up with similar platforms. In the future, -the simplicity of porting and its cost will be one of the leading criterias on defining what is the best technology available (nowadays i.e porting between 2 platforms with Unity3D can be as simple as one click and a bit of beforehand-planning = very low cost and large platform coverage) -the workflow iteration time and usability will be equally important as a game done with one technology of same quality but cheaper than a game done with another technology will make the former a winner middleware -the amount of features available will always be a convincing factor for developers to adopt a technology. While in practise, the quality of the features is worth far more than their quantity, technolgies will need to stay on top of marketing with brand new ”next-gen” mind blowing features (which will in most of cases never be used at their best in the context of a game) In conclusion, engines will focus on making sure their competitor does not have a clear competitive advantage, they will keep using their reputation as one of their main selling point and they will keep adding new features, most of them mostly for visibility purpose and they will focus more and more on usability to reduce development costs. That said, each technology approaches each problem a different way, so the developers should be able to pick what fits the best for their game depending on how the technology solves their problem and of course on the licensing prices.

Company B
Software Architecture and the Creative Team
In what ways are the creative team allowed to contribute to the design of the software architecture?
The technical team discusses all the time with the creative team in order to find the most optimal ways to implement and expose certain features. They influence the design of the software by communicating what workflow would work best for them before the tools get implemented.
Which features in particular do your software suite provide to help the creative team do their job?
If by software suite you mean middleware, Unity 3D helps us by providing a very intuitive graphical user interface, flattening the learning curve for creative people put in contact with the tech for the first time. It also allows the tech team to fully customize the tools available to create content and extends the existing one. -Customizing the editor is the most helpful feature -the built-in asset importers allows the art team to test their assets in-game without the help of a programmer
To what extent do the creative team use the game engine and its features to try out new ideas?
Some members of the creative team just use the game engine to import the assets and test them in-game, some others with a very light programming background take a copy of the project and start messing around with features implemented by the tech team. They prototype their own features such as camera controls, simple gameplay mechanics by doing minor changes to the base code. Once they have something they are happy with, the tech team makes the code production quality and the feature stays in the game. Of course, the level design is entirely done in the Unity editor, so all level related work is done in the game engine editor, this meaning placing objects in the world, triggers, level flow and tweaking gameplay metrics. Once a level is functional, the art team will iterate on it by replacing placeholder assets by the proper ones, tweak the lighting environment and make the level look pretty in general. While this does not always involve new ideas, most of the time, it turns out great ideas emerge from the ability to try to envision their original idea directly within a level.

Company B
Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?
This question is partially answer by the previous ones but as a summary, yes, it is part of the creative team's routine. This applies both to prototype phase for rapid prototyping and for production phase for level editing.
Implementing Changes
How would your company reason about implementing the above mentioned change?
We would start with a meeting involving both artists, level designers and programmers (or at least one acting lead to represent each department). During this meeting, we would establish at which extents does an earthquake alter the world. E.g does it only move around certain objects or does it deform the terrain heights, destructs part of the environment and so on, and we would come to an agreement on how to create the most dynamic result with the least amount of work and considering all possible known technical limitations (does it involve network play, does it require a lot more new content or can it be automated by code...). At this point, everyone is in sync on what is expected from this feature, programming and art will meet to agree on art assets necessary for prototyping (if there is a need for prototype art), art will then get started on those and programming will start a quick prototype (1 to 2 days at most) in order to have a visual starting point for the next phase. During the next phase, programming and design will meet to give feedback on the prototype and iterate to get the system to give the result as close to what is expected by design. Once the behaviour is as expected, programming and design will discuss on usability, what parameters can be controlled and how to expose them in the most simplistic way possible, which data needs to be serialized in order to save/reload the state, or can we get away by ignoring certain data. Once in agreement, programming will fix the prototype code to be production quality and will expose the parameters as discussed. The feature goes online, it will eventually be iterated again in the future within reason if an great idea pops up and makes sense to implement.

Company B
Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?
The way we iterated the construction of this feature, the raw idea brought by the creative team was early on made more realistic with the minimal quality loss possible by discussing with programming, then the idea was polished and made functional as a game feature doable within a realistic schedule during the meeting with design. Such a feature would have been estimated from 3 to 5 days from the first meeting to the point it is production ready. Management would have supervised the process at a key timing, and would generally not have intervened in the decision unless the final time estimate is beyond schedule or budget.

D.2 Company D

Table D.2: Company D's Questionnaire Results

Company D	
Game Engine	
Which game engine do you use?	Custom
Did you use any middleware or third party modules in your game?	
No	
Do you have any thoughts regarding the evolution of game engines in the future?	
No response	
Software Architecture and the Creative Team	
In what ways are the creative team allowed to contribute to the design of the software architecture?	
We're a small company, so our creative team is our development team. We call all of the shots.	
Which features in particular do your software suite provide to help the creative team do their job?	
We strive to make it easy to author new content and ensure that it functions as desired.	
To what extent do the creative team use the game engine and its features to try out new ideas?	
Fully - the creative team is the same as the development team. If we can implement something, we can try it out.	
Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?	
Yes.	
Implementing Changes	
How would your company reason about implementing the above mentioned change?	
We'd debate what it would take to make the changes and whether or not it was worth it with respect to the primary goal: shipping a fun game.	
Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?	
Everyone would be involved equally.	

D.3 Company E

Table D.3: Company E's Questionnaire Results

Company E	
Game Engine	
Which game engine do you use?	Several, Unity3D i smuch in use
Did you use any middleware or third party modules in your game?	
Yes (please specify): Several depending on project. I'm not on top of all the details.	
Do you have any thoughts regarding the evolution of game engines in the future?	
No response	
Software Architecture and the Creative Team	
In what ways are the creative team allowed to contribute to the design of the software architecture?	
As long as it is in line with recommendations from the devs, the creative team contributes quite a bit.	
Which features in particular do your software suite provide to help the creative team do their job?	
primarily in rapid prototyping, tracking of tasks and milestones and in concept / document sharing	
To what extent do the creative team use the game engine and its features to try out new ideas?	
Not so much. The creatives usually mock up in other software of choice. The devs then hands of prototypes, or pair programs with creatives	
Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?	
In close cooperation with the devs, yes.	
Implementing Changes	
How would your company reason about implementing the above mentioned change?	
The creative team would request input from the devs regarding physics, since the devs are likely to find realistic sources for implementation. Based on the feed back, the creatives would probably look at the realistic constraints vs. the goal of the system, and provide a specification for prototyping by the devs. <return> :)	

Company E
Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?
All. In general terms, the management always have the last say. Still, the example provided gives me an idea that it has been decided that this shall be done. So the creatives and the devs will be charged with finding the solution. The management needs to sign off on any budgets, and preferably get some alternatives to select from.

D.4 Company H

Table D.4: Company H's Questionnaire Results

Company H	
Game Engine	
Which game engine do you use?	Various
Did you use any middleware or third party modules in your game?	
Yes (please specify): Native mobile apps = Unity Mobile browser apps = HTML5 internal engine with a few smaller javascript shims and libraries Facebook pc = Flash Including a wide variety of libraries, ranging from 3D with away 3d, physics with box2d and smaller tweening engines and similar.	
Do you have any thoughts regarding the evolution of game engines in the future?	
My main focus when it comes to game engines is related to the cross platform and cross domain nature of game development. Future engines, as the trend seems to be proving, should run across mobile/desktop/console and seamlessly be able to run logic on client or server in same language to gain effeciences in production. Important that a game engine these days is no longer just rendering/scengraph/scripting with tools, it needs to provide, or be a component of, a more complete set of infrastructure given the more online and persisent nature of games	
Software Architecture and the Creative Team	
In what ways are the creative team allowed to contribute to the design of the software architecture?	
The creative team works with the techincal team to find the best way for the content to fit with the architechture of the software. Be it what level of flexibility should be exposed to the tools, how the tools should interact with development/integration or live versions of the game service. To simple things like finding the optimal workflow and matching that with choices in technology.	

Company H
Which features in particular do your software suite provide to help the creative team do their job?
at the most high level the software suites that manage to expose usable tools to create assets and content that allow the designers to directly work with the runtime and game experience are very helpful. a simple example for smaller flash games is the ability to create animations and UIs indenpendat from changing any line of code in a way where the artist is in control and can work autonomously and effeciently without needing to go throuh any hoops to test out features.
To what extent do the creative team use the game engine and its features to try out new ideas?
We aim as much as possible to allow the creative team to use the game engine, or subparts of the engine to allow them to do rapid prototypes, having the games quest/scripting engine as flexible as possible so they can mix and match various triggers, events and data to create new types of experiences they want. Naturally, the more data and tools driven we can be the more the creative team can exerpiment.
Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?
yes, however we do tend to create "Feature oriented teams" of 3 people, 1 coder, 1 artist and 1 designer, that work in union centered around a feature so that all disciplines are represented as a single unit rather than splitting creative/tech teams too much. This works well as our teams as small and relatively low techincal complexity
Implementing Changes
How would your company reason about implementing the above mentioned change?
1. Why are we doing this? What is in it for the player? What metrics will this drive for us if any? 2. How long time will it take? Will it cause any knock on effect or future work other than this single change? Will it distract from other critical features? 3. Do we think it will be awesome fun? ps. as a note, i would be dissapointed if this type of feature change became a big discussion. If the engine is written well, and if i understand the underlying example this should not be a major issue to do. (having done something exactly like this in [one of our games] using Box2D). I would just imagine the static game world still being a polygon geometry that could be modified.

Company H
Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?
We strive to have features and design be as bottom up driven as possible, if work added to the project is large and more than 2 weeks of work we need to have a discussion as a team to consider how it fits in with overall priorities. If the team itself cannot reach a conclusion together it is up to the Producer to make final decision. The Producer can go to management and explain the rationale and get a more executive decision back if it is a major shift in time or budget. However in this case, like i said, i would preferr the team just to go ahead and hopefully have made good enough engine/tech and be fast enough team to execute on that without much further ado

D.5 Company M

Table D.5: Company M's Questionnaire Results

Company M	
Game Engine	
Which game engine do you use?	Corona
Did you use any middleware or third party modules in your game?	
No	
Do you have any thoughts regarding the evolution of game engines in the future?	
More GUI based engines for "standard" game types, where the visuals and the story is more important. But there will also be a need for open engines that will let the creative and talented create new and innovative games. Looking at the increase in Game developers worldwide, more engines will evolve into "easy to use" products as more money is invested in them. Example of this is unity 3d engine.	
Software Architecture and the Creative Team	
In what ways are the creative team allowed to contribute to the design of the software architecture?	
In our experience quite a lot, as we have always been a small team, and that in turn demands us to focus on streamlining the production flow, enabling the creatives to actively use the softwares most effectively.	
Which features in particular do your software suite provide to help the creative team do their job?	
Our internally created programs used in production, lets the creatives plug them directly into their commercial tools, in order to test and view assets directly in the game. And that is a feature that we fin vey effective, as what you see if what you actually get, and any problems can be addressed directly.	
To what extent do the creative team use the game engine and its features to try out new ideas?	
See above.	
Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?	
Yes indeed, see above again please :)	

Company M
Implementing Changes
How would your company reason about implementing the above mentioned change?
No response
Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?
No response

D.6 Company Z

Table D.6: Company Z's Questionnaire Results

Company Z	
Game Engine	
Which game engine do you use?	Our own
Did you use any middleware or third party modules in your game?	
Yes (please specify): libvorbis DirectX 9	
Do you have any thoughts regarding the evolution of game engines in the future?	
No.	
Software Architecture and the Creative Team	
In what ways are the creative team allowed to contribute to the design of the software architecture?	
No response	
Which features in particular do your software suite provide to help the creative team do their job?	
No response	
To what extent do the creative team use the game engine and its features to try out new ideas?	
No response	
Is using the features of the game engine part of the creative team's routine, as in doing a sort of rapid prototyping?	
No response	
Implementing Changes	
How would your company reason about implementing the above mentioned change?	
Look, if we decide to make a game, it is because we find the experience to be had in that game meaningful and/or important. Therefore we execute whatever technology is required to make the game work. Sometimes compromises are made because the amount of work required to make things happen would be disproportionate, but these compromises never strike to the heart of the game.	
Between the creative team, the technical team, and management, who will be involved in this decision, and how important will their opinions be?	
I make all major decisions like this. It's pretty much unilateral.	