PRESENTING A METHOD FOR GAME DEVELOPMENT

playable|design

MASTER THESIS
Media Technology and Games - Design and Analysis
IT-University of Copenhagen
March 2007

Supervisors: Espen Aarseth
& Claus Seeberg Friis

Written by
Frank Wisnes
Bo Behrmann Jensen
Thorvald Kingbo
Simon Larsen

IT University of Copenhagen

Media Technology and Games - Design & Analysis

IT-University of Copenhagen
Rued Langgaards vej 7
2300 København S.
Tlf. 72185000
www.itu.dk

Title: Playable Design

Abstract:

The computer game industry is currently facing problems on how to structure and manage a production due to the cause of ever growing budgets and manpower required. Methods from other fields have been tried. However, these are not built with the purpose of developing games and do not facilitate the entire production or qualities of a game. We have during the course of the project addressed this matter by devising a method of our own which should enhance the process of creating games. A deeper analysis of current computer game developments identified the main problems to occur in the pre-production which suggests that this is where the point of insertion should happen. We have based our solution on theory from fields such as software engineering, innovation theory and game design theory and supplied these by observation made through experiments where we simulated parts of a game production. The result is the EVE method, which stands for *Experimentation*, *Visualization* and *Evaluation* and is based on principles of creative process control, flexible design in form of prototypes and easily accessible feedback through early testing.

Bo Behrmann Jensen

Simon Larsen

Frank Wisnes

Thorvald Kingbo

Supervisors: Espen Aarseth & Claus Seeberg Friis
Circulation: 4 copies
Page Count: 212
Project period: September 1, 2006 – March 1, 2007

1 Table of contents

2 Foreword

During the last decades computer games have experienced a rapid growth and have gone from being part of a subculture to being an accepted way to entertain you. The request for bigger, more complex and of course funnier games has created an industry which tries to translate the demands of the consumer into innovative design that can entertain for hours and hours, no matter the costs. A common analogy is to compare the game industry with the movie industry and Hollywood, where movies have gone through the same development, as games currently are doing, meaning an expansion in both complexity and budget. The difference between the two media is that movies are an established field with well known theories and guidelines of how to create a good movie. Since computer game theory is a fairly new field the point of agreement has yet to come - scientists is still discussing how to analyze games and what characteristics that makes a good game. This makes the whole situation surrounding game development all the more difficult. If you are to decide whether or not to invest millions of dollars into a game, one could imagine that you would be interested in knowing if the concept would attract buyers and thereby being worth investing in. This problem area is what we have tried to explore during the course of this project. We have devised a method which can help developers and investors to obtain the appropriate knowledge to make a qualified decision on whether or not an idea is worth pursuing. The method supplies a framework and tools from which game designers can test both new ideas and improve already existing ones. To form a solid foundation for our method we have found inspiration from industries which share similarities with the game industry. Furthermore the method is based on observations from experiments performed to validate theory and our own supplements developed during the project.

2.1 Readership

The project was developed during the fall/ winter semester of 2006-2007 as part of the Master degree in "Media Technology and Games - Analysis and Design" at the IT-University of Copenhagen and was handed in March 1, 2007. The report is aimed at game developers and companies who want to improve or change the way they currently are handling development of games. The report assumes that the readers have a general knowledge of games and to some extent are familiar with software development. The report is divided into three main parts; an analysis of the evolution within the computer game industry based on information from post mortem articles and interviews with employees at Io Interactive, a theoretical perspective where known theories on the topic is examined along with findings made through our own observations, and finally "The EVE method", where we introduce our suggestion of how to approach game development.

During the report we have chosen to include case studies from post mortem articles in the report in order to exemplify and substantiate the theory presented. These will be separated from the text and be placed in text-boxes along with a brief explanation to how and why it fits the theory.

2.2 Acknowledgement

As part of the project we have worked closely together with the game development company Io Interactive, located in Copenhagen, and would like to thank them for opening their door and helping us during the course of the project. We would particular like to thank Thomas Howalt, Jesper Donnis and Mads Prahm, all from Io Interactive, for going beyond the call of duty and sharing thoughts and ideas on the topics addressed in the report. Furthermore we would like to thank Helle Marijnissen, Jeremy C. Petreman and

all the other people who provided us with information about the work procedures used at Io Interactive.

3 Introduction

Like Christopher Columbus leaving the Spanish port of Palos, we too start a journey towards uncharted waters. Columbus set out to find a western route to the promised Orient in hope of prosperity and acknowledgement, and even though he failed in his quest, it led to one of the greatest discoveries in history. He thought he knew what he was going to find on the other side of the horizon, but to his surprise he found a new and perhaps better treasure - America. Like the quest of Columbus the uncertainty of game design is both exciting and terrifying, and when examined closer you find more similarities between these. One could say that a game designer faces the journey of Columbus every day - to sail into waters where no one has yet dared to go. The job of a game designer is to set out with a blank map, hoping to find new lands to explore, before he reaches the edge of the world. Luckily Columbus had his compass to guide his way. Game designers are not so fortunate to have a tool to keep them on a right track at all times, when searching for new ideas. This is what we hope to change with this project. To create a method which helps to chart unfamiliar territory and at the same time create a "map", which helps to find the undiscovered treasures; new game ideas and knowledge surrounding these.

3.1 Problem area

> *That was the very centre of his genius - he invented things that anyone could have thought of, and men who can invent things that anyone could have thought of are very rare men.* ***-Pratchett 2001, p. 37***

The quote by Terry Pratchett is really the essence of every creative process - some people have the ability to invent things that we all think, we could have thought of the moment we see them. The brilliant ideas which shape the generations to come are often the simplest ones, both in appearances and usability. We all know the situation where we think "why didn't I think of this?", but the harsh reality is that most of us do not have the ability to see the potential in the little things. Can we learn it? Maybe - the truth is that some people have a natural talent for generating ideas, but maybe with the right tools more had the chance to become a great architect, artist or game designer. Imagine if we could devise a method which could help us shape and evaluate our ideas. This could give us a foundation to decide if it is worth investing time and money into.

Every industry handles their creative process in different ways based on years of experience with various approaches. Take a look at the software industry for example, where many different methods have evolved to ensure quality and usability of the end products. From big and complex methods and frameworks to simple and fixed guidelines; they all try to standardize the process from initial idea to final release in order to ensure that the customer receives a usable product. Some methods have been used since the dawn of computer science while new ones emerge constantly according to changes in society, e.g. working conditions, know-how in the general population and political decisions. A branch of the software development, which over the last decade has evolved from a small enclosed industry to a billion dollar indus-

try with thousands of people employed, is the computer game business. The explosion in numbers of people playing games have brought an enormous amount of capital into this business, and by doing so creating larger development cycles and teams, which again requires the companies to develop methods, similar as those the traditional software industry is using, to handle this new situation. The need for such methods is only increasing and at this point it is hard to find one which can be adopted directly for computer game development. So why not just change and adapt the methods already developed for traditional software to the gaming industry? The mismatch between the preconditions for ordinary software compared to those of a computer game makes many of the methods unusable in the context of a game. Traditional software is built on the concepts of functionality and usability - the users must be able to do what they want in a simple and unmistakable way - whereas a computer game's most valuable asset is what is known as the *fun factor* - a game must be fun to play. This makes the two kinds of software somewhat different and requires that we handle them in different ways. The preconditions of traditional software are both measurable while it quickly becomes hard to measure whether or not something is fun.

Does this mean we cannot develop a method which takes the inherent quality of a game into account? No, it does not - we just have to build a method which is based on this and not solely those of traditional software. During this project we will devise a method from which games can be developed with the viewpoint on how a game works. We will stand on the shoulders of others, meaning that we will not develop a whole new method but instead use proven practices from other methods and mix it together with observation done through our own experiments. Our main focus will be on the design and conceptualization process where our intention is to provide tools to handle the process by visualizing and evaluating a concept. This should give

a developer the possibility to scrap bad concepts early on in the development and thereby save time and money. We wish create a way of working which raise the chance to become a "genius" like the person, Terry Pratchett describes.

3.1.1 Problem statement

This project will outline a method based on proven practices which can enhance and support the process of creating, exploring and evaluating ideas in the context of game development.

3.1.2 Project scope

During this section we will try to clarify how we intend to devise our method and what we hope to achieve by doing so. The method we suggest will contain three elements - *Experimentation*, *Visualization* and *Evaluation* - where each stage will help a game designer to create his vision and thereby making him able to communicate this to others. The idea behind our method is that it can work in multiple phases of the development process but will be formulated with the intention to be used in a specific situation, namely the design and conceptualization process. Our focus is to create a way of working with game design, which hopefully *can* be used in other situations as well, but for the sake of simplicity we will limit the scope and focus to this area.

A computer game is a composition of many elements that requires different design approaches. To avoid making a method which aims to solve everything and end up solving nothing we have chosen to divide the components of a game into three sub-categories and then to focus on one of these. The three categories are:

- Game mechanics - rule-set, controls, actions, behaviors within a game context.
- Setting - background story, theme, universe, scenario, setting surrounding the game.
- Art - visual and audio elements presented by the game to the player.

A more elaborate explanation of these concepts will be presented in chapter 8. The categories can be seen as a trinity where each improve or amplify the others; however, it is our opinion that they are of different importance. We believe that the essential part of any game is the game mechanics. Why? Badly designed mechanics leaves a player with a poor experience - if you do not do the job right when designing these you risk designing a game no one likes to play. In addition, if the story and art is prioritized over mechanics we might as well call it a technical demonstration, interactive story or something else, because the values that make it a game is lost - a game will not suddenly become good just because a story or extravagant graphics is added. One could say that the core of the game is the game mechanics while the other elements only help to amplify the player experience. Therefore we have chosen to aim our method to handle game mechanical issues that facilitate the gameplay experience, consequently we will not elaborate on how to make a story, create animations or elements such as these.

The method is intended to be used by developers and companies to strengthen the game design process. However since we have limited our scope to this part of a game development, we feel that it is important to explain how our method fits game development in more general terms. Therefore we will also take a closer look at methods currently used in the game industry and software industry and afterwards elaborate on how we adapt a method to this scenario. Besides the practical approach we also hope this project will add to the ongoing discussion in both the academic world as well as the gam-

ing industry on how to create fun games within the boundaries of time and money.

3.2 Project overview

During this section we will clarify how we intend to proceed with the project and how our report will be structured. In order to understand the context in which our method should work we will make an analysis of how the game industry is currently working. The analysis is divided into three sections – first, a broader introduction to computer game development; what is it and how to it relates to other design approaches. Secondly an analysis of post-mortems[1], where we will examine how known games have been developed, and try to understand the pros and cons of the way different companies approached their respective developments. Lastly Io Interactive has given us the opportunity to explore and understand their way of working. Therefore we will perform a series of interviews with people in leading roles at Io Interactive, which hopefully, along with the post-mortem analysis, should give us a picture of how, where and why problems occur in a production.

Besides game theory we will study other fields which have similarities with game development and hopefully get inspiration for how we can frame a method to fit a design process when creating games - one could say we want to map known design theories to game development. We will examine fields such as Innovation and Design, Software Engineering, Organization theory and Interaction Design. Furthermore, in order to validate our method we will perform a series of experiments. These should give us an idea of how to map the different tools and techniques used, in relationship to each other. The experiments will be a combination of groups of one, two or four persons where

[1] For more information on post mortems refer to page 39 in chapter 5.2.

the purpose is to develop a small prototype by using concepts from theory and our own ideas or supplements, in a development cycle of one week. We will carry out 6 of these experiments where each week will present a new theme to each person or group. The development cycle will include everything from idea and design, implementation, and play testing phase. It should resemble the situation we intend our method to be used in and at the same time give us the possibility to extract how and what will work in a game development process. Lastly, and in order to answer our problem statement, we will present our suggestion of how a design process can be structured, based on our analysis and experiments, in order to fit and make sense in a game development context.

4 Game development from the outside

This chapter will take a look at the current situation in the game industry, and give a brief overview of its development from small scale to large scale productions; looking at how development methods have changed and how the industry views itself and its own methods. We will also take a look at the development within the software industry, and briefly compare the two fields. But first we will look at some hard facts from the industry.

4.1 Facts about the industry

Game development is becoming ever more expensive, and while the time span of the productions has not changed much, they include more and more people. While definite numbers backing up these claims are hard to find, post mortem articles in *Game Developer* and on the website Gamasutra.com is one possible source of facts. A post mortem is an article written after a project describing the project from a "what went right", "what went wrong" point of view. These articles include development time, budget, and the number of full time developers for each of the projects. Using this as our source of information, we will over the next few pages compare game productions over the last 8 years[2] and look at the direction in which the industry is moving. It is worth noting that these numbers have been collected after the completion of the project, which means that a game being in development between 2000 and 2002 with a release date in 2002, will show up in the statistics as a 2002 entry. Appendix B lists the entire source material for the graphs presented on the following pages.

[2] From March 1998 to November 2006.

4.1.1 Development time

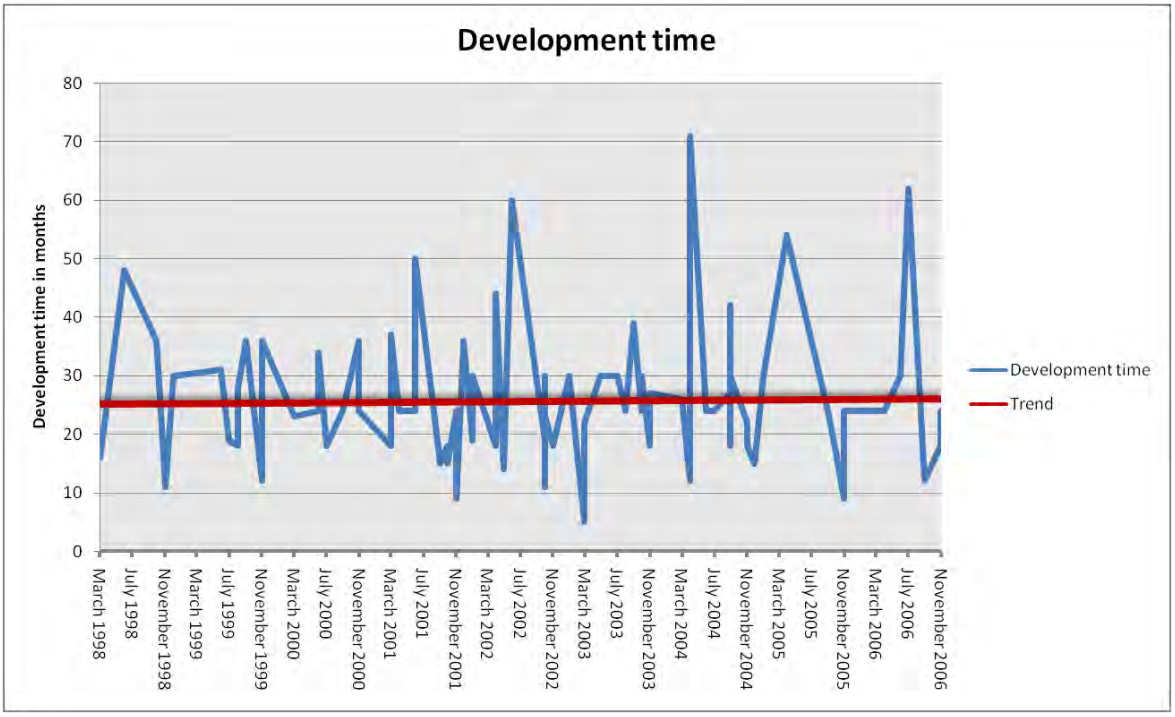

Figure 1: The average development time in months.

The average development time of productions seem to be rather consistent. There are different games in the analysis that took very little time and very long time to develop, but the average production time is stable at around 25 months. In other words, roughly two years from concept to finish game on the store shelves. The interesting thing to observe here is the thing that the graph is not showing; *how* the time is spent. Some productions go from the conceptual phase to full production in a matter of months and others spend 12-18 months in pre-production before moving onto full production. Reasons for this vary greatly; for some it a mere matter of trying to nail the concept that leaves them in the pre-production phase for longer than average, and for others the game they are producing is a sequel in an already firmly established franchise, that requires less pre-production which means the team is more likely to move to full production earlier.

4.1.2 Budget

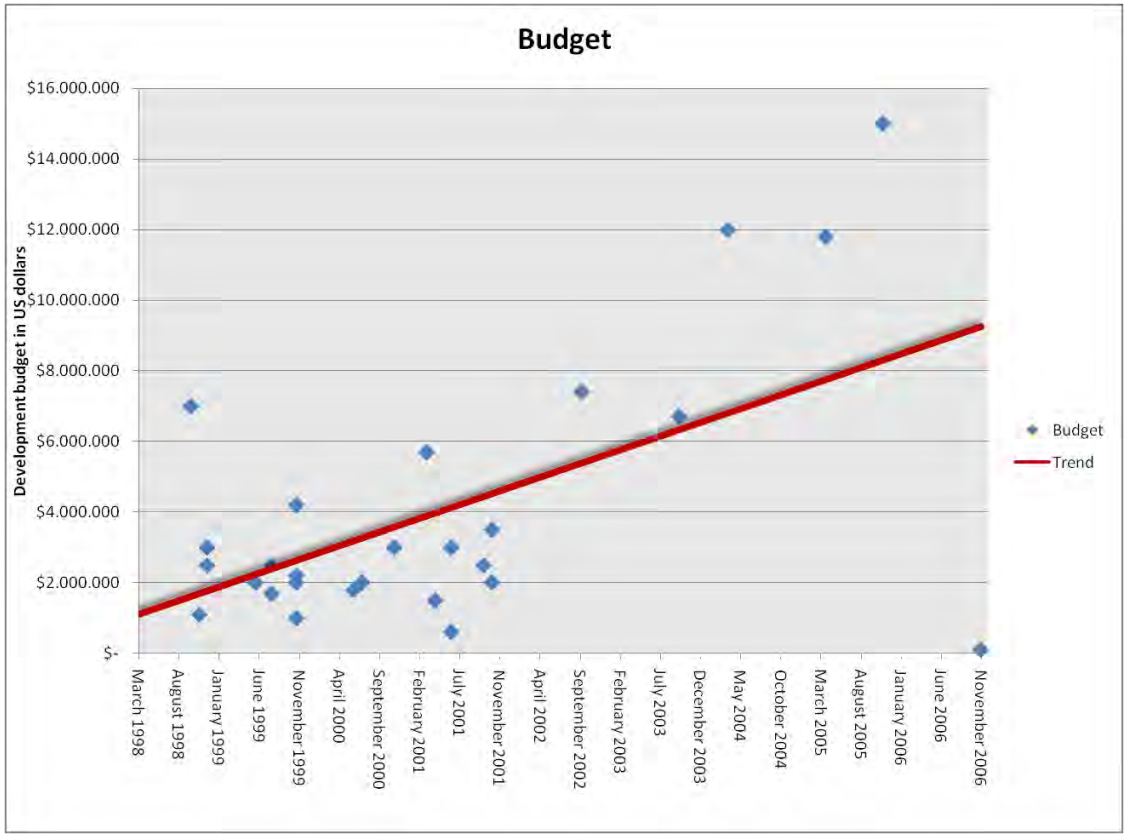

Figure 2: The production budget in millions of dollars.

More interesting is it to look at the average development budget. Unfortunately the information about the production budgets was not always available in the postmortem articles. But since the trend is so clear that we must assume this is directive for the rest of the productions. From 1998 to 2001 the overall budgets seems be somewhat stable around 2.5 million dollars per production[3], and then the budget goes into a steadily rise from 2001 and onwards to end up with an average budget of no less than 13.5 million dollars in 2006. That is a rise of no less than 540%. Games are becoming more and more complex both graphically and technically, and for companies to be able to

[3] It goes from 2 to 4 million dollars, but on average it is 2,59 million dollars for the entire period

compete with the general marked they ramp up the number of people working on the production in order to complete the game in time. That is, of course, if that is the parameters the developers choose to compete on. There is a major problem with this development. The prices of the games have stayed the same over this entire period, and if inflation is factored in the prices can be said to have gone down. While more games are being sold today then 5 years ago, this increase is nowhere near the increased cost of development.

4.1.3 Team size

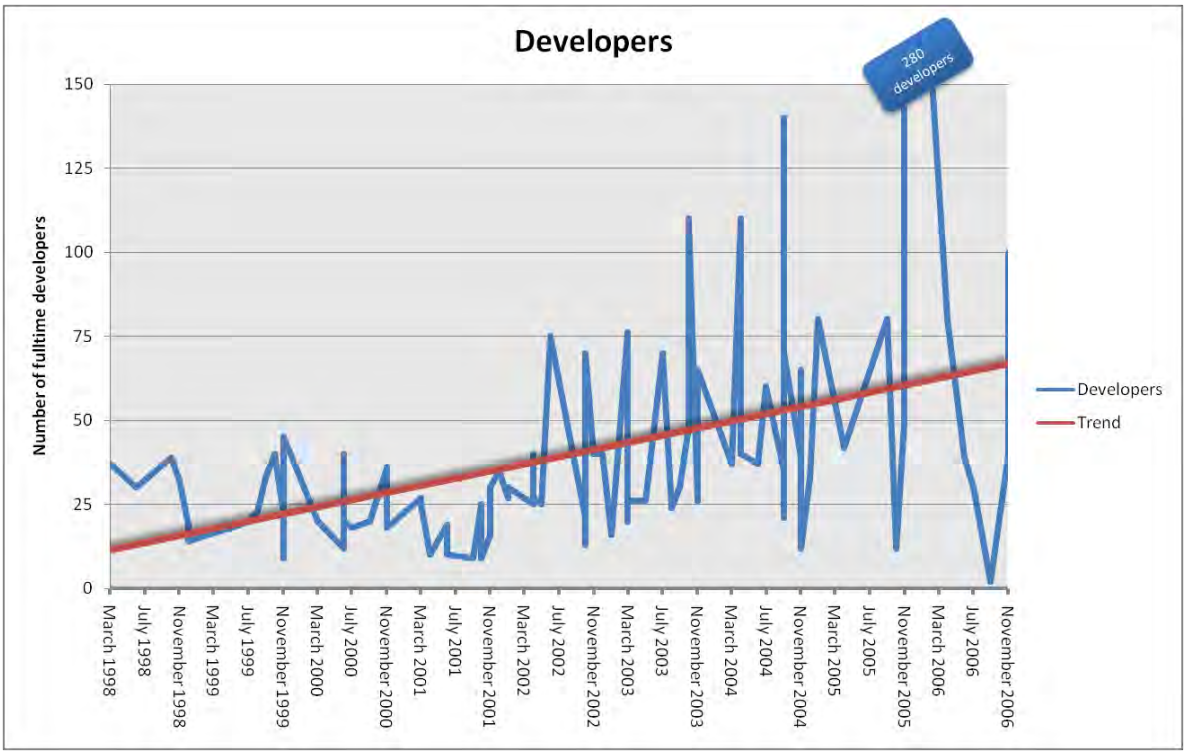

Figure 3: Number of full-time developers on each production.

A similar trend can be seen in the steadily increasing number of developers working full-time on the productions. Seeing as the single most cost consuming element of game development is wages, this comes as no surprise. The sudden increase in the number of developers just around 2001 might be con-

nected with the release of the Playstation 2 the previous year, indicating that new consoles and more advanced technology requires a larger development team. *Peter Jackson's King Kong* pull the average number of developers rather high in late 2005 since it had no less than 280 full-time developers working on it at Ubisoft. But the fact that more developers are working on each production remains intact, and whether or not this trend will continue remains to be seen. While you would think that there is an upper limit as to how many developers will fit in a single production, *King Kong* shows how that limit is higher than 280.

4.1.4 Conclusion

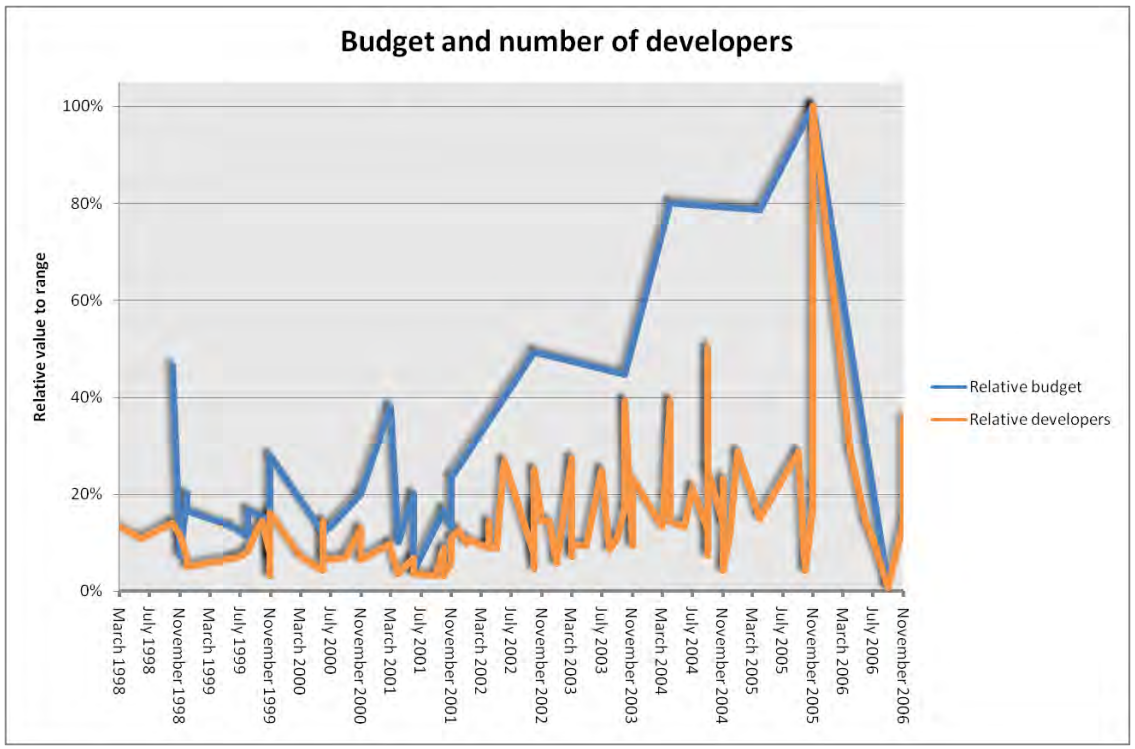

Figure 4: Comparing the number of developers and the production budget.

The trend is clear when the two graphs are combined. The graph above clearly shows where the money is being spent in game development. Hardly sur-

prising material, since almost all of game development is about the knowledge and skills the individuals' team members brings to the productions.

Since the numbers themselves do not reveal much about the day-to-day productions, all we can conclude from this is that game development is indeed becoming more expensive because of the rise in the number of people working on each project. However, the revenue does not scale at the same rate as the development cost increases, meaning that "throwing more people at the problem" will soon stop being a viable economical solution to increasingly more complex productions.

4.2 Game Production

Even with all this, game development as a field is still in its childhood. While the first commercial games are over 35 years old[4], the industry were long dominated by a small group of dedicated individuals. They were the self-taught, first-generation game designers, building games alone or in the company of a few friends while laying the groundwork for the multi-million industry we have today. As time passed and the game development teams grew from a single person to teams of several hundred people, so did the design and development setting change.

Lately, some of the veterans of the industry have given their suggestions as to how game development should be done, either through lectures, books, or articles like the ones found on the website Gamasutra.com and in Game Developer Magazine. Even so, the majority of the literature on games concerns itself with game design more than game development. So let us start off by

[4] The first successful commercial game was Atari's Pong, released in 1972.

drawing the distinction between game *design* and game *development* before we do anything else.

4.2.1 Game Design versus Game Development

If we were to draw a line in the sand, one way to do it would be to say that game design is a part of game development, while game development is the entire process from initial idea to the final product. This is not entirely true, but it works as a black and white categorization. One could say that the two are completely separate, but again it is all a question of practical application. Andrew Rollings and Ernest Adams (2003) have little trouble focusing purely on game design in their book, while Julian Gold (2004) on the other hand almost completely ignores the design aspect when he talks about game development. Books on the subject are either focused on design or development, sometimes giving the other field a few brief pages. Game design has been given far more attention than game development and with good reason; while designing games is a matter of looking at what has been done in games before and how this can be done in new and interesting ways, game development is something else, something far more businesslike. It involves the financial side of things where investors and the publisher want to have a say, and there is little room for artistic creativity here. But let us start by taking a closer look at the different views on game design.

4.2.2 Game Design

Ernest Adams and Andrew Rollings (2003, p. 4) describe game design as the process of imagining a game; defining the way it works, describing the elements that make up the game, and transmitting that information to the team that will build the game. While being rather general, it does offer some in-

sight into the relation between design and development. The game design is the vision on which the game is built, and it describes the game in detail. What the design does not do, is explain the underlying code structures, describe how the team will go about building the game, rate features in order of importance, or in any other way explain how the game is going to be built. And while the designers dictate how a game should work and what elements it should contain, the actual *building* of the game is left for someone else. How these builders do their job is entirely up to them, and is not something the game designer writing the design usually worries about.

This way of dealing with a design phase is different from the way the design process is done in traditional software development. Here, the software designer writes flow charts to show the relation between objects and classes, while the game designer writes charts to describe the interaction between characters, units, and other actors in the game world. These two things looks the same on the surface, but the comparison is only skin deep. Game design is done on a higher level without focusing on the underlying code structures. This can be seen in how theory on the field does not take the job of the programmer into consideration, silently promoting a division between design and implementation. However, one of those who do consider this distinction a problem is Julian Gold (2004, p. 384), who suggests the addition of a technical designer to the design team. This designer would be the person who turns the ideas expressed in the design document into the flow charts we see in other software development structures.

4.2.3 Game Development

There are few sources of information on how to do a full game development, and Cerny's Method[5] (Cerny & John, 2002) is the closest there is to an actual method for developing games. This method gives a macro view of the entire production from idea to finished product, and tries to deal with some of the common problems we see in game development today. After having identified what they see as myths in game development, they go on to present the way they create games. The result is the following process:

Preproduction

- Document: Macro design
  - 3 Cs: Character, Camera and Control
  - Visual style
  - Completed key technology
- First playable

Production

1. Micro design
2. Play testing

Their method focuses on the preproduction phase of a game project, the goal of which is to gather as much information about the project as possible. At this stage the game design should be on a macro level, including only the three Cs, the visual style, and complete key technology to be used in the project.

[5] While developed by both Mark Cerny and Michael John, it is commonly referred to as "Cerny's Method" or simply "The Method".

This way of doing a preproduction is all about taking chances early on, figuring out what works while exploring the limits of the technology, and getting concrete information as to how long elements like levels will take to build. At the end of the preproduction phase the team will have a short but detailed micro design document where all the relevant knowledge learned in the preproduction phase is documented. This document should include information about the character and how it moves, exotic mechanics, level structure, level size, level content, and overall structure of the game. This document is complemented by the *first playable*. The first playable is similar to a "vertical slice"; a large prototype showing all the key features in the game compressed into one to two fully playable levels of publishable quality.

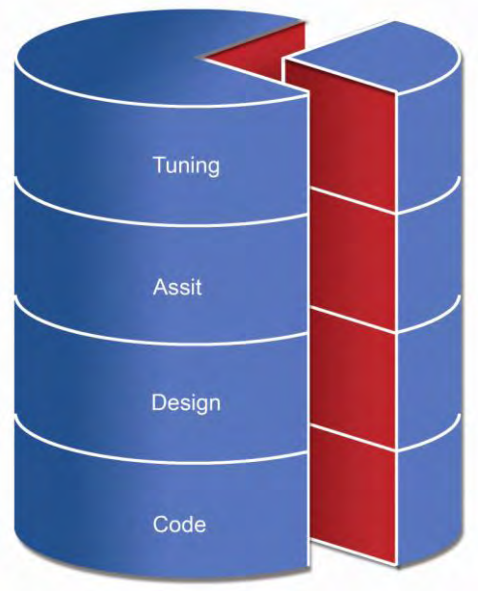

Figure 5: A vertical slice of a entire game.

Aside from Cerny, writers focus most of their time on game design without describing a full production. *On Game Design* (Rollings & Adams, 2003) and *Game Design Workshop - Designing, Prototype and Playtesting Games* (Fullerton, Swain, & Hoffman, 2004) are examples of books in which game design is the only focus. In addition to Cerny, Tim Ryan (1999) touches on how to do the actual implementation. In his article, he focuses on how to make the design phase more structured, but this is only a small piece of the entire production.

4.3 Software Development

The most obvious place to start when trying to find a methodology suitable for game development is the field of software development. While the end

user experience is not the same in games as in traditional software, the work environment in which the products are created can be said to be the same. Both are built by people with the same skills, using the same tools and programming languages. A game has more focus on the visual and auditory elements, but as soon as you strip away the artistic side of things, game development can easily be compared to software development. While game development can be said to be software development with an added element of creativity, it is software development nonetheless.

The fact that software engineering focuses on functionality and usability while caring little for the fun aspect of the end product, has to be taken into consideration. What these methods do is that they simply help the project manager manage the team's time and available resources in order to create the best possible product. Compared to the game industry, the software developers have had more time to refine the methods in which to accomplish this. With little documentation of the current methods being used in game development, looking to software development is a natural next step.

4.3.1 The Waterfall Method

Traditional software development often used the Waterfall method originally proposed by Winston Royce in his article “Managing the Development of Large Software Systems” (1970). Ironically, the method Royce proposes is not as rigid as it later became known as. He actually recommends doing some of the steps twice;

> If the computer program in question is being developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar critical design/operations areas are concerned (Royce, 1970, p. 334).

Unfortunately, his method was adapted in the software development industry as a *single-pass* method, where once one phase was completed iterating back was unheard of. Nevertheless, even though Royce suggested to "do it twice" the waterfall method fails to address the high-risk elements of development in a constructive way. High-risk areas such as implementation and usability is pushed until the end of the development cycle, leave the impact and cost of changes much higher.

Even so, the version of the waterfall method described by McConnell (1996, p. 136) is one of the best known development methods in the software world. It is a lifecycle model, which means that a project following this method goes through a set of predefined phases described by the method. Each phase ends with a review which purpose is to see if the project is ready to progress to the next phase. In the waterfall method a project starts off with an initial concept, followed by a detailed requirement analysis identifying exactly what the software should be able to do. Then follows the system and software design phase, where the overall architectural design of the code structures is done. Once the design is in place, the actual programming begins. This means that all the design is done *up front*, before any of the actual implementation starts.

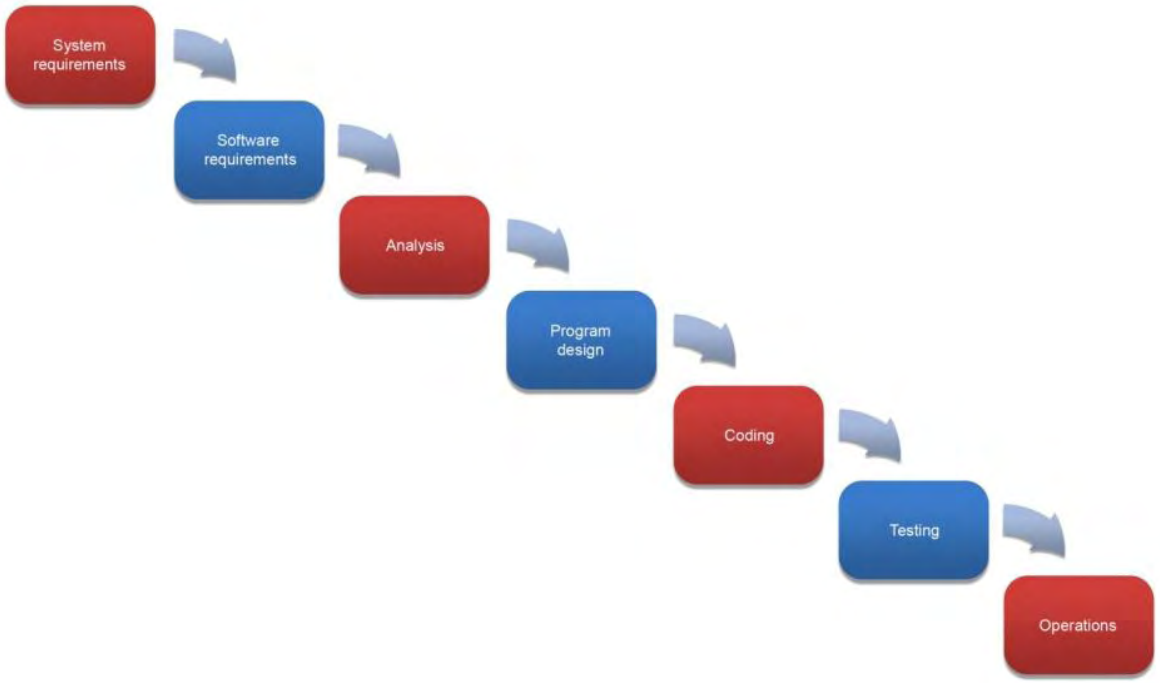

Figure 6: The waterfall method

This way of seeing a software production fits very well with game design theory and the idea that a game project starts off with the game designer writing a detailed design document before anything else happens. The different phases do not overlap, and going back to a previous phase is possible but according to McConnell it is not recommended. The method takes as a premise that the initial design is the right one; going back due to a flaw means retrofitting the fix into your old design while putting the rest of the project on hold. This is because the design and requirement analysis are done right at the start of the project, and herein lies the problem with the method. It only works if you know *exactly* what you are building and are familiar with the technology you plan on using when building it. If this is the case the method brings order and structure to the project, and the lack of flexibility will not be an issue because you know what you are building and how. However, this is not the case for most software productions and never for large scale game projects. With most of these products, the end user can have a hard

time specifying exactly what he or she wants until the final product is ready to be tested. In this situation, a more flexible method needs to be considered, one which allows for continuous design and user feedback throughout the span of the project.

4.3.2 Spiral Development

Another traditional software engineering method is *spiral development*. This method incorporates risk management as a driving force throughout the project. As described by Sommerville (2001, p. 53) the development process is represented as a spiral, with each loop in the spiral representing a phase in the project. Each loop is split into four sectors:

Objective setting	A detailed plan is described containing the objectives for the phase along with constraints on the project, the risk involved in the phase and possible alternative strategies based on those risks.
Risk assessment and reduction	An analysis is made on the base of the risks identified in the first sector, and steps are taken in order to reduce these risks.
Development and validation	Depending on the risk analysis, a development model is chosen that best deals with the kind of risk the project is currently facing.
Planning	The loop is reviewed, and plans are drawn up for a new loop if needed.

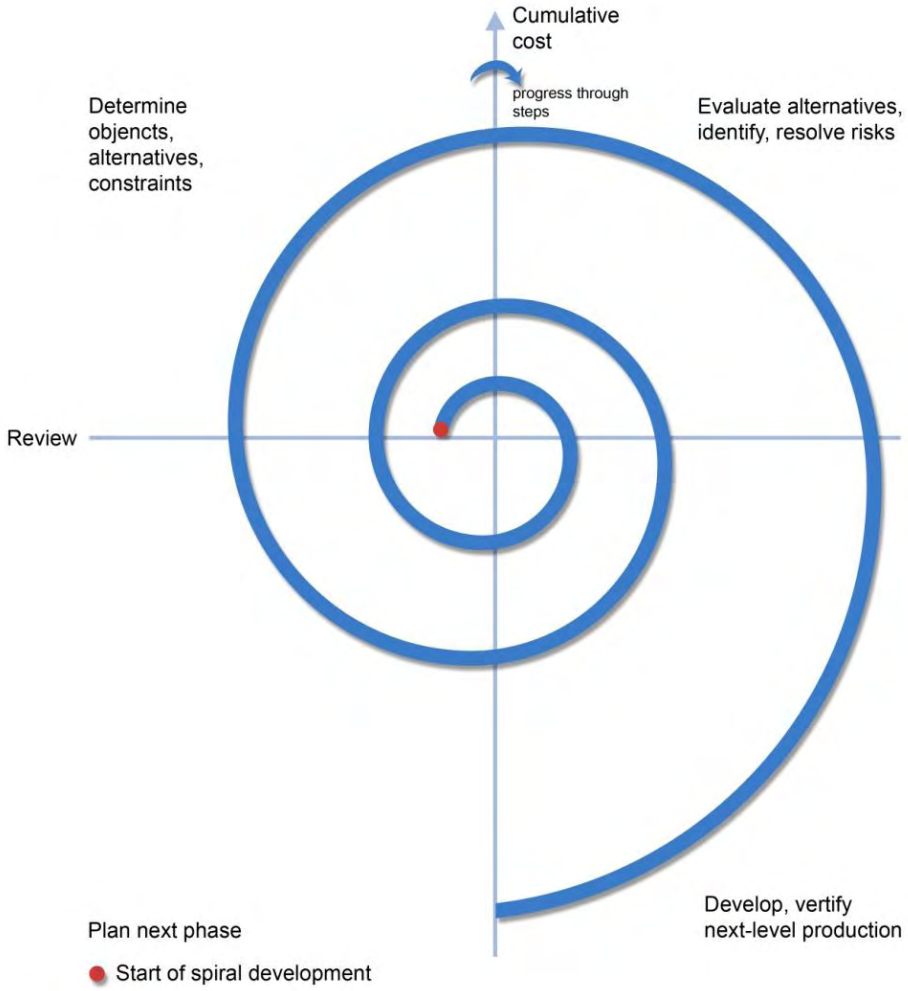

Figure 7: Model of the spiral development method. The project starts in the center of the spiral.

By being able to incorporate almost any development method, spiral development is more of a meta-method when compared to other methods. Since, because in each spiral a different method has to be chosen based on the risks that spiral represents, a thorough understanding of other methods' strengths and weaknesses is a prerequisite for using it. It also requires a team that is willing and able to switch work methods several times throughout a project.

4.3.3 Agile methods

Iterative development methods tackle high-risk areas much earlier in the development (Larman, 2004, p. 58). While spiral development comes close to being iterative in the way it bridges all the different traditional methods together, there is a movement in software development that takes it one step further. The iterative methods do not eliminate the risk, but they do try to cope with it in a more meaningful way by addressing it head-on earlier than most of the traditional methods.

The agile and iterative methods are an attempt to handle the sudden changes and unexpected problems software productions often face in a more adaptive way. The best known agile method is *Extreme Programming* (Beck & Andres 2005).

4.3.4 Extreme Programming

First published in 1999 and then revised in 2005, Kent Beck's book about Extreme Programming (XP) received a lot of publicity in software circles when it first came. It is tag lined "embrace change", and with good reason. What Beck tries to do with XP is to reduce the cost of change by designing and developing the software at the same time by continuously expanding the product through iterations. But more than that, he tries to bring some humanity back into software development and show that productivity does not equal burning out people through overtime.

XP is built around five values, and based on these values Beck draws a set of principles and practices. The practices are the realization of values, and the principles work as a bridge tying the two together. The five values are *communication, simplicity, feedback, courage and respect*. Communication because only through proper communications will a team be able to draw on each

other's strengths and avoid doing the same thing twice. Simplicity because you should always ask yourself "what is the simplest thing that could possibly work?" (Beck & Andres, 2005, p. 18). Get feedback in order to spot mistakes and adapt to rapid change. In addition, you also need courage to throw away things that do not work, and the courage to tell your co-workers your opinion. Overarching all these values is respect for the rest of the team, as people and as professionals.

While the values are guidelines for how members of the team should relate to their fellow team mates as well as the project as a whole, the principles are an understanding of these values on a more practical level. They lie somewhere between ideology and actual practices, not as abstract as the values yet not as tangible as the principles. Beck lists 14 principles for XP[6], and together they form the framework for the practices.

Based on this, Beck suggests a range of different concrete practices to be used throughout the project. Practices are day to day activities of the XP team. The core practice in XP is short iterations with weekly releases and continuous design paired with feedback from co-workers and customers. Through this, sudden changes in requirements will be less of a problem because the developers do not suffer from bad plans made in the beginning of the project. Beck also promotes the idea of a customer as a permanent member of the team, providing constant feedback on the product being developed. By adapting to change and planning for change, the situation of the programmer becomes more stable and less stressful than if change were suddenly placed upon him with no prior warning or mechanisms for dealing with them.

[6] The 14 principles are *humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps,* and *accepted responsibility*.

4.4 Fun factor

When comparing game development with software development, it is important to keep in mind the difference between games and traditional software. Software is designed to be functional, and is judged based solely on how well it solves the problem at hand. Can the word processor do a good job of setting up a document? Does the image editor produce good images? What we want from our software depends on what kind of software it is, and we want different things from our internet chat clients then we want from our music editors. With games on the other hand, the user demand is easy; they want to be entertained. This leads us into an entirely different setting, and instead of asking if the program does what it is suppose to do, the question is simply if it is fun. But it is not that simple. Games have expectations as well. A horror game is expected to provide a certain experience, which is completely different from a sports game. Just like nobody opens *Word* to look at their pictures or *Photoshop* to write a document, players do not start up *Doom III* for a good laugh or *The Sims* for fast-paced action.

The difference, however, is the approach taken by the user. Traditional software are designed as tools, games are designed as entertainment. What kind of tool you need determine what program you use, and what kind of entertainment you want determine what games you play. Software has to solve the problem it is designed to solve, but this can be tested purely on a technical level. As far as software development is concerned, there is no need to include the user before the usability test. With games the same two elements apply, and while the usability issues are much the same, the "problem" a game is set to solve is much more complex. Games are trying to entertain through an element of fun, and fun is a much more subjective thing then the value of a tool. While it is possible to come close to an objective evaluation of the value of a digital music player and a word processor, it is hard to narrow

down the element of fun in a similar matter. Fun is all-important when it comes to the success of a game, and while there is much to be learned from software development, this should not be forgotten.

5 Game Development from the inside

In the last chapter we had a look at different views on game design and game development, and compared the two terms in the light of theory. We also had a look at software development theory from the traditional waterfall method to the agile methods represented by XP. In this chapter we will take a closer look at how these development methods relate to game development, and compare the two in the light of the problems they are meant to solve and the results they are intended to produce. In doing this we will also briefly touch on the discussion around documentation which currently exists in both game and software development. The chapter concludes with an analysis of the game industry through findings from the post mortem articles from Gamasutra.com and Game Developer magazine.

5.1 Big Design Up Front

The waterfall method promotes the idea that designing everything before you start implementing anything is a good way to develop software. And in some cases it is. The software industry did it for years, and looking at the current discussion on how to write a game design document it seems as if the game industry is doing it as well. In the previous chapter we looked at methods originating from the waterfall and the agile way of thinking, and now it is time to compare the two in the light of game development.

5.1.1 Waterfall versus Agile

Say you are making a Flash game[7] for a website. It is an adaptation of classic poker, complete with the network code that will allow several people to play against each other. The framework for the poker game is already in place and a website portraying a classical casino has been up and running for some time. The team creating the poker game has already created both blackjack and baccarat using the same tools, and has a good understanding of how to go about implementing a new game.

This is close to the perfect scenario for applying the waterfall method. The designer knows exactly what the game is about; poker is a known game and in a casino setting variations from the original rules will not be tolerated. The team making the game has experience from making similar games in the past, and they know from previous experience the possibilities and restrictions of the tools they will be using. Designing everything before bringing in the programmers creates few problems in this scenario, and would be the preferred way of doing it. While a setup like this is possible in other sections of the game industry besides casual games, it is a rare occurrence and most likely tied to an expansion pack of, or a sequel to an existing game.

In all these cases the designers know what they want to give the player and the players know what they want. In the first scenario, the players want to play poker online. How it is wrapped and sold is in the hands of someone else then the development team working on the implementation, and whether or not the players enjoy playing poker is something the designers do not have to worry about. The same goes for the sequel and the expansion pack; with the tools in place and the end user defined, a company can always

[7] A game made using Adobe Flash is commonly referred to as a "Flash game". Adobe Flash is a tool for making browser based applications, and is a favorite among many developers of casual games.

choose to play it safe and release a new product offering more of the same with few new additions.

However, most game productions are not like these. Designers want to bring something new to the table; the *unique selling point* (USP[8]) which distinguishes their game from their competitor's and impresses the investors. There is also a good chance that the game relies on technology the programmers are not familiar with. Even a genre which has been tried and tested by others can be a substantial challenge. Implementing car handling in games is still hard to do right even if people have been doing it for years, and the number of technically poor first-person shooters we have seen suggests that even *that* genre can offer some serious challenges. Chances are, though, that a game designer wants new and compelling features which also include unfamiliar technical solutions.

Whether it is the design or the implementation that ends up causing problems does not matter, either one makes designing everything up front a risky proposition. As far as software development goes, Clements and Parnas (2004) list the following problems with doing up front design in traditional software development:

- The clients or users are not sure what they want.
- They have difficulty stating all they want and know.
- Many details of what they want will only be revealed during development.
- The details are overwhelmingly complex for people.
- As they see the product develop, they change their mind.

[8] The simple fact that this is a coined term in game development circles indicates how much emphasis is put on this. The dictionary at GameDev.net defines USP as "*Unique Selling Points. Normally what will be put on the back of a box or an advertisement showing how a game is different and better than its competitors and predecessors*".

- External forces (such as competitor's product or service) lead to changes or enhancements in requests. (Clements & Parnas, in Larman, 2004, p. 5)

Looking at this list, it becomes clear that the problems software development have been facing for decades are very similar the gaming industry's current problems. The players do not really know what they want; they know a good game when they play it. The details of exactly what they want only become apparent to the developers during testing. This also ties back to Cerny's dismissal of the focus group as only being able to tell you "what not to do" (Cerny & John, 2002), turning the idea of the focus group up-side-down. The game designers on the other hand have a problem with explaining the game design perfectly the first time, and ends up changing the design as they see the game develop. And finally, other games coming on the marked during development can end up having a major impact on a game being developed which is suddenly found to be lacking what every gamer now takes for granted. The lesson to take from this is that game development and software development are not as different as game developers often wish to see them. Both industries are fighting many of the same issues, whether they are making games or security systems. The game industry has a lot to learn from software development when it comes to development methods. Another problem area which is shared by both camps, is one concerning the role of documentation.

5.1.2 The Design Document

The discussion around waterfall and agile methods leaves the idea of the extensive game design document with some problems. Tim Ryan (1999) and many with him, have advocated the idea of the detailed 100-300 pages design

document as the best way to go about designing a game. Rollings & Adams (2003) also include a section about how to go about writing your game design document, listing Chris Tailor's[9] template as a possible starting point.

There are many good reasons for starting a project with a large game document. First of all, it is cheap. A single person can produce a game document single handedly without using more expensive tools then a stack of papers and a pencil, and though the process of writing it he will be forced to consider elements of his concept on a different level then when it was just an idea floating around in his head. However, expecting this document to give an accurate description of the entire game is expecting too much. Even a team of people writing the document together will not make much difference unless it is the equivalent of the casino poker game they are making. Still, a game design document is something many publishers value (Rollings & Adams, 2003, p. 586), an important point which should not be forgotten. There is a lot of politics in the game industry, and some of it is tied directly to the design document.

While a large document does have its uses, the problems with the waterfall method suggest that a document like this should not be used as a bible for an entire production. For that, developers are better off looking to what the agile methods propose and make a document which follows the micro/ macro design mentality taken by Cerny & John (2002). Create a macro document of a few pages containing just enough to get started, and then make the first prototype. This fits well with Becks idea about *Incremental Design* (2005, p. 51), suggesting that "the most effective time to design is in the light of experience". Of course, he is talking about the design of a code base, but the de-

[9] Chris Tailor is the designer behind *Total Annihilation*, *Fallout: Tactics* and *Dungeon Siege*. His template for creating design documents can be found here: www.designersnotebook.com/ ctaylordesign.zip.

sign principle is the same as for Cerny. *Design once you know what you should be designing*, and find a way to figure this out before you have invested a year in the paper design. Cerny have advocated one way of doing this, and later in this report we will present a different way.

5.1.3 Burnout

Another reason for looking at different development methods is the current turnover rate in the industry. According to IGDA's[10] Quality of Life white paper from 2004, people in the game industry complain about crunch[11] and overtime, and the statistics say that 51,2% of the people asked do not see themselves working anywhere in the game industry 10 years from now. "For the industry as a whole, such a high turnover rate is nothing short of catastrophic, and it goes a long way towards explaining our difficulty in ensuring that our projects run smoothly" (IGDA, 2004, p 17.). It is clear that something has to be done, and if the agile and iterative way of thinking helped software development out of their problems, it might just be that it can help game development out of theirs. An example of development practices can be found in the informal article EA: The Human Story[12], in which the spouse of an EA employee tells the story of an EA development team. This article describes the inhuman work conditions for a development team at EA Games, and created a storm in online circles from people in similar situations. In the aftermath of this the software engineers won a $14.9 million settlement from

[10] IGDA is an organization for game developers worldwide, formed to promote and strengthen the game industry as well as improve communication between professionals. For more information see www.igda.org.

[11] Working day and night to meet a deadline.

[12] An article written by the spouse of an EA Games employee, creating a strong focus on the work conditions in the industry. The original article can be found here: http:/ / ea-spouse.livejournal.com/ 274.html.

Electronic Arts, and the artists won a similar $15.6 million settlement[13]. This goes to show that even if the number of people who want to get in to the game industry may seem endless, you can only push the ones already working there so far before they revolt.

5.2 Post Mortems

While most companies like to keep their competition at arm's length, one way of getting an insight into actual game development practices is through post mortems. These articles provide us with a unique glance into the world of game development through the eyes of the developers themselves, and even though they might not be completely honest, the articles still gives us a good indication of what the people in the industry list as good and bad practices. Most of these articles also include numbers on production cost, team size and development time, and in the last chapter this was used to outline some concrete facts about the industry. The following section is a quantitative approach to the post mortems, where we will look at what the industry thinks about itself. A thorough study of these post mortems would need an entire report, so we have chosen to focus on the most extreme cases.

5.2.1 Flexibility

Flexibility is a topic which is raised in many of the post mortems, and there are several ways in which projects can be said to be flexible. One is in relation to the toolset the developers use when making the game which ease the flow of assets between the technical programmers, sound programmers, graphical artists, and level designers. Another is flexibility when it comes to the design

[13] http:/ / www.gamasutra.com/ php-bin/ news_index.php?story=6747

process as a whole. Many companies use their own engine when making their games, and some companies like Valve and Epic even sell their engine to other companies and offer it for free use to mod communities[14].

As far as tools go, some developers had flexible tools, others saw their project suffer under the lack of flexibility and wished their tools were more flexible. The developers of *Tropico* (2001) were very happy with their unit and building editor and call it "invaluable for balancing and tweaking" (Smith, 2001), and the level designers on *SWAT 3* (2000) say that the use of the flexible level editor Worldcraft[15] instead of 3D Studio Max saved them "a ton of time" (Saladino, 1999). At the other end of the spectrum, *Startopia* (2001) needed to have all its models, animations and objects coded manually in order for them to work properly, leading to an excessive use of programmer resources while leaving the graphical artists with little to do (Imlash, 2001). *Diablo II* (2000) had much of the same problems, causing a lot of extra work and leaving sound engineers "painstakingly creating .AVI[16] movie versions of animations in order to synch sounds with actions" instead of simply creating a tool for them in the engine so that other than programmers could create content within the game engine (Schaefer, 2000).

While lacking some technical tools, *Diablo II* is on the other hand a good example of a game where the designers had a flexible approach to their design. They never had an official design document, and only made a rough plan before they started experimenting with new ideas (Schaefer, 2000). Granted, the game is a sequel to the original *Diablo* (1996) which was a huge success, but this could just as easily have made them sit down and draw up everything

[14] The word "mod" comes from "modification", and refers to a modification made to an existing game. Games like Half-Life and Unreal Tournament have large communities of fans dedicated to creating mods.

[15] The level editor for Half-Life.

[16] Audio Video Interleave, a multimedia container format for windows.

they wanted to improve before starting to build the game. Instead they designed the game continuously as the project progressed, and to great success[17]. This is similar to the process used by Naughty Dog, the company behind the *Jak and Daxter* and *Crash Bandicoot* series. Working closely with Mark Cerny left its mark, and they are "making large-scale games and shipping them on time" through an iterative and agile development method. By using a "flexible, macro-level scheduling scheme" their schedule is more accurate, and slips in the schedule can be handled on a case to case basis to help keep the production on track (White, 2002).

However, as far as documentation goes there are mixed experiences. Having a flexible design should not be confused with having too little design or no design at all. Like Beck and Andres (2005) stress in their XP methodology; design enough to get you started then keep on designing throughout the project. *Fallout: Tactics* (2001) is an example of a game which suffered from this lack of design. With no clear vision behind the game and not enough work put into the pre-production, people outside of the design team ended up doing a lot of the design work during implementation. The playable demo which came out of that process "absolutely stunk", as they put it, and only with the help of their CEO did they manage to get the game back on track (Oakden, 2001).

5.2.2 Documentation

Documentation seems to be a tricky business for many developers, and too much is just as bad as too little. *Big Mutha Truckers* (2003) suffered from over documentation, causing two major problems. First, the sheer size of the de-

[17] Diablo II received a MetaScore (metacritic.com) of 88 and the reception from the players where huge. Due to the well designed online part the game is still being played today, 6 years after the original release, and has a substantial fan base.

sign document made it difficult extract information from it, and the fact that all the relevant information about an element of the game often spread throughout the entire document did not make it any better. And second, the fact that it was written as an in-house marketing tool for selling the concepts meant that "instead of concentrating on the 'hows' and 'whys' of the game's production, it was instead focused on the 'whos' and the 'wheres'" (Jobling, 2003). The developers behind *Trespasser* (1998) had the same problem, but in a somewhat different form. Even after they went into production, the only document describing the gameplay was "a prose-based walkthrough of what the main character would do as she went through the game, and a short design proposal listing the keys which would be used and some rough ideas of what gameplay might actually be" (Wyckoff, 1999). While it is possible to enter a production with a document like that as a starting point, it requires a toolset like the one offered by the agile methods in order to pull it off.

The problem with many of the productions is that they start out with an extensive design document and expect it to cover every aspect of the game. It rarely does. When the teams behind games like *Fallout: Tactics, Command & Conquer: Tiberian Sun* (1999) and *Big Mutha Trukers* say their document were lacking, it is a simple answer to a more complex problem. These problems are identical to the problems software development have been facing, and simply writing more will not solve this problem. In some cases *less is more,* as with Cerny's minimalistic approach to documentation. And if we are to take a lesson from the pitfalls of software development, more would be just more, giving little to the project other than the illusion of certainty. But no matter how you look at it, some documentation is required, and the key to writing this documentation seems to reside in the pre-production.

5.2.3 Pre-production and the vision

Regardless of how a developer chooses to go about creating his game, the development process will always include a pre-production phase of some sort. The goal of the pre-production is to create a plan for the production, be it a strict or more open-ended schedule. While this plan is the overall goal of the pre-production, there are several other elements which also need to be considered. The most important of these, is the *grand vision*.

That came to be the major roadblock for the developers behind *Soldier of Fortune* (2001). During early development the team changed from thinking they were making a first person shooter to thinking it was a team-based tactical shooter, and this lack of a vision made the game hard to sell to their publisher. A lot of work ended up being wasted, and in the end "the SoF team learned the hard way that a day of preplanning saves a week of rework" (Biessman & Johnson, 2000). The developers behind *Tropico* had somewhat of a different problem. Having just finished making *Railroad Tycoon II* (1998), they went from doing a sequel to developing a brand new idea from scratch. The lack of a proper pre-production led to the people on the team having different visions about what the game was supposed to be, and this became a growing problem throughout the production as these differences surfaced. Only through a painful process did they even manage to complete the game, and it took its toll on the team. In the end, "working on Tropico stopped being a passion and became just a job for many on the team, leading to low morale and loss of productivity" (Smith, 2001).

It is a common problem in this phase to fill the game document with features without looking at the game as a whole. While this is understandable seeing as the game is not yet made, it still proved to be a big problem for many developers. As each designer proposes features based on his understanding of

what should be the core aspect of the game, it becomes harder to narrow down the vision later on.

Dungeon Siege (2002) is an example of a game suffering under extreme ambition. The pre-production left the team with a long list of features they wanted to implement, but the lack of a core concept made it hard to cut in the feature list. In the end, they crunched for two years to get all the features in simply because they did not know what to do. “We didn't crunch to make up for lost time, we crunched out of uncertainty“(Kijanka, 2002).

The team behind *Thief* (1998) also had the problem of filling their design with features during pre-production. It had multiplayer modes, branching mission structures, tools players could combine to create new tools, and more. However, as the production progressed they realized their scope was too wide, and “these and other "cool ideas" were correctly discarded”. Instead they focused on the core aspect of the game; a linear, mission based, single player game based around stealth. By dropping the multiplayer support and focusing on implementing player tools which worked within the interface of the engine they were using, the team could keep a stronger focus on the stealth part of the game (Leonard, 1999). The result was the start of the stealth genre as we know it today.

Generally speaking, there seems to be a problem in the pre-production when it comes to game projects. Even when people make an effort to keep their design at a realistic level they end up either having to cut features or prolonging their production, and the cost of doing so depends on whether or not the team considered this during pre-production and planned their production accordingly. Getting everyone in on the same vision is another big challenge, and defining the core aspect of the game is also seen as very hard. Making sure everyone on the team understands the vision and the core of the game seems to be one of the many keys to success in game development. But even

with that in place, there is still a question of whether or not the idea is a good one in the first place.

5.2.4 New ideas

Another challenge in the pre-production phase is to figure out if your idea is good. Regardless of the game company's relation to their publisher, getting a bad project back on track or simply shutting it down early and before a lot of money have been invested is in everyone's best interest. The Cerny method shows one way of testing concepts early in the development, and his method is being used by some productions, most notably *Spider-Man 2* (2004) and *Ratchet & Clank* (2002). Regardless of the method used, the idea of having something playable is valued by many developers.

The developers of *Drakan 2* (2002) got to experience this the hard way. Due to a lack of early gameplay testing, they "implemented systems and built whole levels before the team realized that something was never going to work from a gameplay standpoint" (Denman, 2000). *Soldier of Fortune* had this same problem, and the lack of a playable prototype after a year of development left their publisher "a little nervous". This uneasiness where shared with the developers, and caused major turmoil as they tried to narrow down exactly what kind of game they were making (Biessman & Johnson, 2000). The actual costs of these late discoveries are hard to measure, but wasted man hours, frustration and de-motivation on the part of the development team are some of the effects.

On the other hand, developers who did get the time to experiment with their ideas early and got something playable up and running were very satisfied with the results. Maxis has long been doing rapid prototyping when they start on a new game project, and on *The Sims 2* (2004) they "used early proto-

types to resolve look and feel issues, to help understand the key emotional connection, and most importantly, to test out the new gameplay concepts". These prototypes where made in the concept and pre-production phase, and in addition to sparking creativity when defining the core aspect they were also a great help to both graphic artists and programmers throughout the production (Bradshaw, 2005). The design process of Ensemble Studios is somewhat different. The design process they used when making *Age of Mythology* (2002) was basically "to get the game playable early and then tweak it until its fun". That way you can identify flawed ideas which sound good on paper but do not work when you add players to the mix (Fischer & Street, 2003).

5.3 Lessons

Perhaps the biggest lesson to take away from these post mortems is *be flexible*. And that works on every level. Those who did not make flexible tools for their animators and sound designers because they thought the project was almost over, ended up regretting this later as things started to draw out. Designers who thought their paper designs were accurate were proved wrong again and again, and the problems this caused were directly dependent on how early in the process these problems were discovered. Looking at all the post mortems, there seems to be some general problems with the early phases of the project. Those who do most of their design on paper and then proceed directly to production are having several problems as the project progresses. On the other hand, there are those teams who know how to do a good pre-production; companies like Maxis and Naughty Dog which employs a version of the agile methods to great success. It is no small coincidence that Will Wright and Mark Cerny are sought after speakers in game development circles.

For most companies these lessons are expensive. What does not break them makes them stronger, and the process of writing a post mortem seems to have helped them make some realizations about how to do their next project. If nothing else, the industry seems to be growing up, little by little, as the different companies learn from their own mistakes. The problem though, is that the difficulties in the industry have been the same over the last 10 years, and worse, they are the same difficulties software development has been facing for decades. While some companies are conscious of these issues, others are not. With the gaming industry becoming more cut-throat as development costs and the size of development teams increase, it is time for companies to start learning from past mistakes instead of repeating them.

These post mortems only represent a small part of all the games which have been released over the last few years, not counting all the canceled projects. We can only speculate in the number of undocumented failures in the industry.

6 Case: Io Interactive

In the following chapter we will take a look at the working procedures used at Io Interactive when developing their games. The chapter will be based on interviews conducted with key employees at Io Interactive. We have interviewed both the producer behind several of the *Hitman* games, and game director on Io Interactive's newest game. For the sake of anonymity the persons interviewed have been renamed and will respectively be referred to as P1 and P2. E-mail interviews were made both with the technical producer on *Hitman* (P3) and *Kane & Lynch* (P4) and furthermore with the lead of the tools department[18] (P5).

Facts on Io Interactive:

Founded: September 1998
Company background:

- Created as collaboration between Nordisk Film & TV and Reto-Moto.
- Sold to Eidos Interactive Ltd in April, 2004.
- Eidos Interactive Ltd was in May 2005 acquired by SCi Entertainment Group PLC.

Games Published:

- *Hitman: Codename 47*, November 2000 - 600.000 copies sold (PC)
- *Hitman 2: Silent Assassin*, October 2002 - 3 million copies sold (PC, PS2, XBOX, Game Cube)
- *Freedom Fighters*, October 2003 - 1 million copies sold (PC, PS2, XBOX, Game Cube)
- *Hitman: Contracts*, April 2004 - 1.6 million copies sold (PC, PS2, XBOX)
- *Hitman: Blood Money*, May 2006 - (PC, PS2, XBOX, XBOX 360)

Employees: 170 + freelancers

The interviews were structured in a non-rigid form and the questions asked served merely as conversation topics. The participants interviewed by e-mail were asked to answer a series of questions concerning their role and responsibility in a development process, as well as how they would characterize the development process at Io Interactive. The entire transcript of all the interviews can be found in Appendix A.

[18] The tools department handles all in-house technology used to develop their games, e.g. engine and editors.

6.1 Idea generation

Io Interactive is currently developing games on both a well-known franchise and new concepts which requires a somewhat different approach with regard to idea generation. Working with a known franchise, such as the *Hitman* series, binds the project by conventions and player expectations. The game must preserve the uniqueness of the previous games if it is to attract the fans again. Many of the developers working on the *Hitman* series have previously worked on the other games in the series and by so know the universe and the development process. P1 explained that the process of idea generation would start with a small team. No more than 5 to 6 people would be assigned where their job was to discuss how to improve and update the universe in regards to the game. Furthermore, the purpose was to identify the direction in which the next game should be heading; is it "more of the same" or is new features needed? P3 added that the assessment of features and content are mainly decided through a series of discussions by the designers involved. P1 explained that a pitch document including descriptions of key features and visual style is written and presented to the creative director and CEO of Io Interactive. They, in turn, give their feedback to the concept whereafter the team adjusts and extends the design. The reason why it is done in this matter is due to the cheapness and easiness of changing direction when all the material done only existed in the form of a design document. At this point no real production has commenced which makes it relatively easy to handle changes.

Another development approach was exemplified by a new relatively small development team at Io Interactive. This team, headed by P2, is trying to handle the game development process in a different way compared to the *Hitman* project. The project is based on a new concept which gives more freedom to experiment and explore new ideas than the *Hitman* team. Instead of taking the traditional approach utilized by the *Hitman* production, this pro-

duction started with only two developers who used a lot of time to outline the entire game and the basic setting. The normal process at Io Interactive is to add a rather large number of people to the project once the first outline is established. However, with the new project the opposite was tried where they instead slowly added people to the production. The result is a project which can easily be cancelled if it turns out to be a failure but also a project where the developers have a clear idea of the vision of the game before going into an actual production. P2 also explained that her role would be very different depending on the type of process. A small team would let her focus on the game while in big productions it would be to manage and direct the team.

> I'm trying to have a very helicopter view of the project and just give people directions and try to be the director that we always wanted to have that roll on the project. To give people feedback all the time, keep people motivated. (P2)

P2 further explains how the intranet is used as an important tool to inform people in and outside the team on the ideas, progress and direction of the project. Here is it also possible to write ideas and suggestions that can improve the design, but in the end it is still the game designer that approves any design changes.

6.2 Pre-Production

When entering the pre-production all important features and details are described in a design document by the *Hitman* team. This document includes description on how the game is expected to work and how the story and levels are structured. Furthermore, an important task for the producer in the pre-production is to assign people to specific tasks and estimate how long they will take to complete their task. At this stage the process resembles the waterfall method described in the previous chapter where each phase is com-

pleted in closure whereafter the next phase is commenced. Since *Hitman* is part of a franchise this approach can be useful to preserve consistence through the games.

The new project at Io Interactive takes a very different approach to pre-production than the one used by the *Hitman* team. P2 explains how they try to make it a much more tangible process by implementing much of the content in some form instead of writing it in a document. However, the design document still works as a tool to communicate the idea. The pre-production team is like with the *Hitman* team at relatively small team to begin with but is slowly extended as the need for more man power increases.

6.3 Prototypes

Prototypes at Io Interactive are used in different situations and vary in scale depending on the problems that they are trying to address. According to P3 the *Hitman* team uses prototypes primary developed in the game engine to test uncertain elements. In this way it is possible to integrate the prototype into the final game. P1 explains that the team is currently waiting for new technology to be completed by the in-house tools department before they can initiate the prototyping phase. This has forced them to think of different approaches than usually, e.g. to use the old engine or short animation sequences to visualize new ideas. However, P1 explains that they have refrained themselves from doing this since it is regarded as being duplication of effort, since this work could not be directly integrated into the final game and therefore has to be recreated it in the final production. Instead, the *Hitman* team has chosen to focus more on developing the story line and the setting for the game.

P2 explains that they too develop prototypes directly in the game engine. However, he is open to alternative solutions such as third-party engines. This could be a good way to break the boundaries of the current technology and focus more on the game experience instead, as P2 explains. Regretfully to P2, the team is currently using in-house technology to develop their prototypes. P2 explains the reason for this as:

> [...] a resistances to that from some parts of the company and there's also some resistance in general in using any tools outside the company for prototyping because people believe – and maybe it's true – that if you prototype in something else then you throw it all away basically. (P2)

Nonetheless, P2 and the team are using prototyping as a tool in much greater extend compared to the *Hitman* team. His prototypes are small and focused on solving a specific problem. Partly because the project is new and in need of a more explorative approach to investigate the gameplay elements; there is no previous games developed by the company to retrieve knowledge from compared to the *Hitman* games.

The information gained from this prototyping phase is used to update the design document as the game evolves. P2 explains that the explorative process of doing prototyping helps the entire team to understand the vision and at the same time allow them to give their suggestion on how to improve or change the game design. Furthermore, the prototype process is also the time where all the elements in the game are explored in order to uncover possible risk areas early on, as this will benefit the production in the long run. P2 explains that the publishers too support such process since they too see things in regards to risk – how much money are we gambling and how much can we make?

The first and most obvious use of prototyping is to get knowledge about a specific problem and thereafter how this knowledge can be used in a constructive way -"Maybe I'll revise my initial design based on observations." (P2). A side-effect of attaining new knowledge is a better understanding of the risk involved. Risk management and risk exploration can help facilitate the ideas in the game to a publisher and management.

Even though prototyping is used during both productions it varies in how they prioritize it. *Hitman* seems to use the design document as the primary source to gather and communicate information, while the new project is using the prototypes as a foundation for the rest of the game.

6.4 Conclusion

The main result of the interviews is the fact that Io Interactive does not use a single developing method. Instead, it seems to be depended on the type of project – well-known concepts versus new concepts. The *Hitman* production is based on the experience made through the previous games while the new project requires a lot of experimentation in order to develop the content needed.

Furthermore the *Hitman* team (represented by P1 and P3) use brainstorming in teams to invent and improve new ideas which could fit into the franchise. Prototypes are used, but are focused on technical difficult solutions, where their purpose is to clarify how to incorporate a new technical feature. Small prototypes could be helpful to explain and communicate the vision to the rest of the team, and reduce the risks in general.

P2 use prototypes to explore the core features which are essential for the main storyline. The game design evolves through a series of iteration with proto-

types. These are used as a design tool along with a design document to find and understand the details of the game. The prototypes were also used to communicate the vision to the entire team and to display the progress of the production to the rest of the company.

A change in the mindset of Io Interactive can also be seen through the interviews. The previous games have all been developed by means of big productions with many people working on the same game. Currently, with the new project, they are trying to go in the opposite direction by allowing a small team of developers to explore new concepts in a relatively long pre-production. The result is a production where every member is an active part of the design process and helps to create the content. This is harder for the bigger *Hitman* production, since it is not feasible to have 70-80 people changing and adding in the design. Instead of the members being an active part of the design process a company meeting is scheduled by the lead-developers where every employee has a chance to comment on the ideas. The two approaches creates very different developments, where the *Hitman* team is able to ensure that the new game is in the spirit of the franchise but at the same time creates a top-down management which can overlook the potential ideas from the employees. Furthermore, it can create problems if the management cannot communicate the vision of the game to so many people. This can lead to confusing among the team members if not handled correctly. In contrary, having a small team makes it easy to communicate the idea to everyone but at the same time creates a flood of information since each team member have an opinion in regards to the game. Both approaches have pros and cons which have to be weighted in relation to the type of game developed.

An interesting point which we discovered during the interviews was the lack of process evaluation. Mistakes, such as long crunch periods, exceeded deadlines and too large development teams where all issues which had created

exhausting developments but nonetheless a thorough evaluation of the process had never been done. According to P1 the reason for this was that everyone on the projects knew why these mistakes had occurred. However, it was further explained that many of the same mistakes happened, or at least could happen, in future productions. This lack of evaluation might be one of the most important reasons why each production at Io Interactive is handled in different ways. A structured way of evaluating a process can prevent many of the same mistakes happening again and could furthermore help the company to find a more uniform way of developing their games. To ignore the benefits of process reflection will create a development environment where people have to start all over again when initiating a new process. To hand down the experience from previous developments is the key to form a solid foundation for new projects.

7 Adaptive Game Design

The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

Therefore the most important function that software builders do for their clients is the iterative extraction and refinement of the product requirements. For the truth is, the clients do not know what they want. They usually do not know what questions must be answered, and they almost never have thought of the problem in the detail that must be specified. Even the simple answer—"Make the new software system work like our old manual information-processing system"—is in fact too simple.

Clients never want exactly that. Complex software systems are, moreover, things that act, that move, that work. The dynamics of that action are hard to imagine. So in planning any software activity, it is necessary to allow for an extensive iteration between the client and the designer as part of the system definition.

I would go a step further and assert that it is really impossible for clients, even those working with software engineers, to specify completely, precisely, and correctly the exact requirements of a modern software product before having built and tried some versions of the product they are specifying.

Therefore one of the most promising of the current technological efforts, and one which attacks the essence, not the accidents, of the software problem, is the development of approaches and tools for rapid prototyping of systems as part of the iterative specification of requirements (Brooks, 1975, p. 199-200).

7.1 Designing versus making

30 years later, this is exactly what the game industry is still struggling with. Unsure what to design or how to approach it this paramount phase of game development is often done in a rather haphazard fashion. It is often approached the same way as construction a building is; by adding one element at a time and slowly making the design larger and larger. But as Glenn Ballard states it “This is the ancient distinction between thinking and action,

planning and doing. One operates in the world of thought; the other in the martial world" (2000). Ballard continues to compare the two; one area being *making the recipe* and the other area concerns making the course *based on the recipe.*

Designing	Making
Produces the recipe	Prepares the meal
Quality is realization of purpose	Quality in conformance to require-ments
Variability of outcome is desirable	Variability of outcome is not desirable
Iteration can generate value	Iteration generates waste

Table 1: The difference between designing and making (Ballard, 2000).

The problem with the sequential approach, the "making" approach, is that it leaves little room for iteration and experimentation with the design. In full production this is less of a problem since you often have (or at least *should* have) a good indication of where you are going. But in the design phase the sequential approach is an unusable solution, due to the fact that when gameplay ideas are implemented in a playable form it rarely results in a system *just* as you imagined. Hence adjustments have to be made both on previous implemented elements and future ones. Doing this sequentially is a huge and cumbersome undertaking. Furthermore the fact that cost of development is steadily increasing makes risk-taking less desirable and both publisher and developer tries to create games that they are sure will recoup their high investment. This coupled with high ambition often lead developers down the tried-and-tested path of sequential development. The high ambition can be seen in the large design documents that often accompany the AAA titles[19].

[19] AAA or triple-A titles are a common reference to high budget production that is released to with much marketing on the worldwide market.

The development of the game *Dungeon Siege* almost collapsed under its own ambition and lengthy development cycle. Although the developers state that they were proud of their final game, the process of making it was not one they willingly would go through again (Kijanka, 2002). Preparing food by following a recipe is very sequential; you cannot boil the pasta before putting water in the pot. However, designing should and must break free of these shackles of conformity to become truly interesting and unique. If not, the design runs a high risk of becoming bland and indistinguishable.

In order to avoid or minimize risk the developers must seek out the risk in areas where it is safe; in early experiments, in continuously maintaining many options, and by being more adaptive in the approach to development and design. The risk will still be present but in a much more controllable form.

7.2 Playable design

Our focus have mainly been on pre-production, because it is in this phase that game development differs the most from software development. It is in the *design* of games that you can find the justification to place them in their own realm in the overall landscape of software development.

Our method, as we have named *The EVE Method* (as in Experimentation, Visualization and Evaluation), builds on our strong belief that in order for the gameplay design to become outstanding it has to be approached in a very *flexible* and *simple* way. It has to be approached from many *different angles* and the individual elements have to be *experimented* with extensively. It has to incorporate strong and frequent *feedback loops* and there must be a system in place that can handle this feedback. It has to be *lightweight* and *adaptive* and most important of all the design must be *playable*. By playable we mean that it

is one thing to design something on paper, it is a whole other thing to get the feel and touch of the design by actually playing it. Prototyping has long been the hallmark of the experimental and explorative development processes and we embrace it wholeheartedly, so much so that we wish to take it one step beyond *traditional* prototyping and focus more on what only recently has been referred to as *lightweight prototyping*[20]. With lightweight meaning simple, small and flexible prototypes that can be developed and tested in a very short timeframe.

7.3 Uncanny Valley

We feel that it is important to stress the lightweight part of this since game developers and designer often run the risk of falling into what we call "The Uncanny Valley of Prototyping". Prototypes follows more or less the same path as the know theory of The Uncanny Valley of visual art. The Uncanny Valley term used for describing the odd feel you get when viewing humans that does not look correctly.

Figure 8: The Beatles in Madame Tussauds Wax Museum in London.

[20] Credit for coining the term *Lightweight Prototyping* (as least within game development) must go to developers at game studio Maxis, especially Chaim Gingold and Chris Hecker, who often refer to their approach to developing the upcoming game *Spore* as doing lightweight prototyping. They have, together with Maxis founder Will Wright, given numerous talks and lectures on the development of *Spore* most noticeably on the Game Developers Conference (GDC) in 2005 and 2006.

The term was first used by robotics engineer Masahiro Mori in 1970 in his article by the same name[21]. The figures in Madame Tussauds Wax Museum are a good example of human figures that fall into the Uncanny Valley. Because the human figure and the appearance of fellow members of the human race are so well-known to us only the slightest offset in visual appearance will become notable.

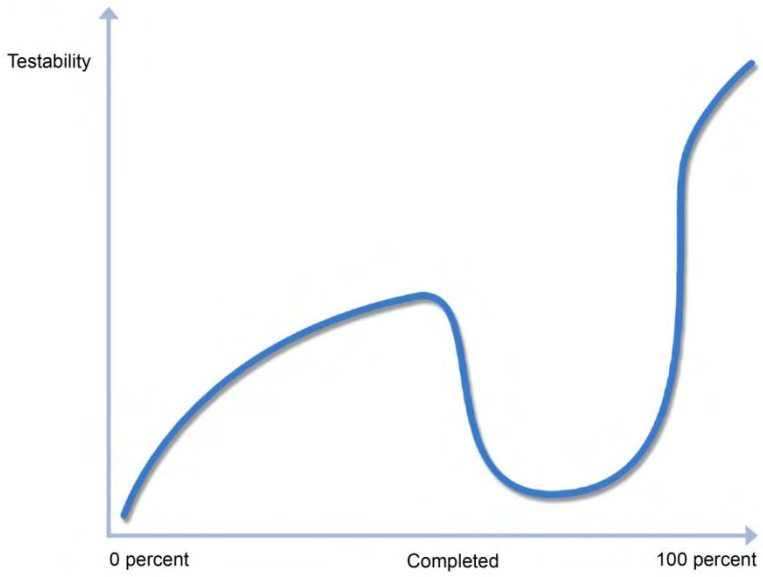

Figure 9: The Uncanny Valley of Prototyping.

If the prototype looks like a finish game when first approached by a tester, it becomes very hard to distract from defects and lacking elements of the prototypes. If the prototype is clearly conveyed to the player "as-is", the person is much more likely to look beyond shortcomings of the implementation and look at what the prototype is really about. You must as a game developer be aware of the large chasm that excites in the uncanny valley of prototyping. It

[21] See http:/ / www.androidscience.com/ theuncannyvalley/ proceedings2005/ uncannyvalley.html.

might take only a few hours or days to reach "the edge" of the valley, but months to fully cross it.

This notion of the Uncanny Valley of Prototyping is certainly something we experience during our own experimentations when we went about testing our prototypes. If the tester were unfamiliar with the normal state of prototypes they often had a really hard time giving valuable feedback. The feedback in these cases often consisted of "why is there no sound?" or "the menu system looks very unfinished". Feedback that was more or less useless since the thing being tested was more in the nature of game mechanics.

7.3.1 The Space Pen

When NASA was commissioned by John F. Kennedy to go to the Moon in 1961 with the famous speech "We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard", they had little idea of hard it actually became. They were facing numerous problems and one of them being that ordinary writing pens did not work in zero-gravity. The astronauts had to be able to write reports on their progress during their missions in space and since computers still were not a viable options for this pen and paper where the logical choice. However, the problem was that NASA was then forced to invent a new type of pen that the astronauts could use in zero-gravity. The regular ones on the market would not function properly in space, since the ink had to "run out" of the pen. A thing that was impossible without gravity. NASA then used years and millions of dollars to develop the famous Space Pen. And what did the Russians

do? They brought a pencil. The result was the same but the Russian approach was simple, effective and to the point. "The simplest answer is usually the correct one", as Occam's razor[22] goes.

The story above is an urban legend and is, although untrue, an excellent example of something extreme complex solved in the simplest way possible[23]. The aim is to make the prototypes lightweight and simple. Make it so that it is both easy to create and easy to understand. Complexity will always enter the product at some point but in the initial design phase it is best kept simple.

7.4 Ready, fire, aim

The important thing about lightweight prototyping is not to create all-encompassing game demos that will ultimately answer all questions about the game design, but to create small quickly developed prototypes that answer small portions. Large demos take long time to develop and often end up becoming smaller scale productions in themselves. The aim is to quickly test out ideas and either fail or succeed, but to do it fast.

The U.S. Marine Corps have what they call the *70 percent solution*. They do not encourage their soldiers to make fast, reckless actions, but they do explicitly state that waiting for all angles to be figured out before venturing forward is equally wrong (Freedman, 2000, p. 5-9). It might not be the perfect solution

[22] Occam's Razor principle was put forward by the 14th-century English logician William of Ockham. The principle states that when multiple competing theories are considered equal in almost all aspects, the theory that introduces the fewest assumptions and postulates the fewest hypothetical entities is usually the best of most correct one. For more on Occam's Razor see Wikipedia.org (http:/ / en.wikipedia.org/ wiki/ Occams_razor).

[23] The real story about the Space Pen is far less interesting than the myth which explains its lower place in the historical hierarchy. The Space Pen was invented by a private company with no funding or endorsement from NASA. Eventually NASA did indeed purchase 400 pens from the company at the price of $2.95 each. Read more about the Space Pen urban legend on Wikipedia.org (http:/ / en.wikipedia.org/ wiki/ Pencil) and on Snopes.com (http:/ / www.snopes.com/ business/ genius/ spacepen.asp).

that you come up with, but there is a very good chance that it is. If you use your acquired knowledge and common sense, waiting for the last 30% of your solution will not make it that much better. It is about just doing it, or as legendary NHL player Wayne Gretzky once said "You miss 100% of the shots you never take".

If you fail often, you can quickly and effortlessly adjust to the feedback, and get back on track with the design. Alternatively failing with a huge and very elaborate design can have both enormous production and morale implications. Recovering from a six month development cycle of a prototype that proved to be unplayable takes time and effort. Recovering from a one-day development of a failed prototype takes just the blink of an eye.

7.5 EVE as a method supplement

The EVE method is a supplement which can be inserted or superimposed on or into existing development methods already in use. The concepts presented herein are meant as mindset changing tools that should raise the development quality and consequently produce a better end product.

Figure 10: The EVE Method can be inserted in all production environments.

7.5.1 The experiments

The background for our method is a series of experiments we did over a three month period during our project. We wanted to get a feeling for what it was like to work with new ideas, and how what kind of problems a developer making prototypes in this manner would encounter. There is no substitute for personal experience, so we tested the theory ourselves. The basic setup for our experiments was as follows:

- One game mechanic developed in one week (5 days)
- A fixed theme (as in a word chosen at random, e.g. “Smash”, “Sticky” or “Change”)
- Change group size: Four members per group, two members per group and one-man groups
- Playtest often

We agreed early on that the game mechanic would be represented by a small game, which should be fully playable at the end of the week. The idea was that the game would rely purely on the mechanic, and should be able to illustrate the mechanic well. The process started off with randomly choosing a word from a list of 40 keywords, and this was done every Friday afternoon so the weekend could be used to find inspiration for game mechanics based on the keyword. Monday started off with a brainstorm around the ideas produced during the weekend, as well as the addition of new ideas that came from the brainstorming process. As soon as each group had narrowed it down to one idea, they got to work on the prototype. Getting something playable and testable up and running as soon as possible was the main priority, and Tuesday, Wednesday and Thursday were spent working on the idea. The deadline was Friday at noon, after which we evaluated the prototypes on

how they compared to the original idea. During these three months we had four evaluation weeks in which we evaluated the process as a whole, and made corrections to our way of working with the ideas.

When we started our experimentations we quickly found that while brainstorming sounds simple on paper, it is a very hard thing to do right. We tried the different techniques described in chapter 8, and as the weeks passed we got better and better at brainstorming. Some words were naturally harder to work with then other words, and not everything sparked the imagination in the same way. For this reason we ended up picking three words for the last few weeks. At this point we were already starting to master the brainstorming process, and the change brought from lifting some of the initial limitations showed that too many constraints that early can be a hindrance.

The actual prototype phase did not see many changes. We often found that the core mechanic of a game changed as we worked and tested the prototype, and at the end of the week the initial idea had changed quite a bit. It shows the classical issue with game and software development in a very small scale; as you see the idea being developed your understanding of the idea changes.

Most of our prototypes where successful in the sense that they illustrated the original mechanic to a level where it could be judged. But the real value in came from the process of working with the idea. Once the week was over a much more accurate description of the idea could be written. It was not necessarily longer than the original description, but it had an undeniably aura of certainty surrounding it. Possible implementation pitfalls were also detailed, and if applicable suggestions of tools a designer tweaking the mechanic later would benefit from.

In the following chapters we present theories from diverse fields, such as software engineering, innovation and testing. Theories that both underline

our method and could be used to solve some of the problems stated in the previous chapters. The theory will be explained in greater detail and as these theories often come from areas other than game development we have tried our best to map and adjust the tools from these theories to fit the needs of game development.

8 Exploration

*An idea that is developed and put into action is more important than an idea that exists only as an idea. – **Buddha***

8.1 Introduction

How do you know if an idea is worth investing time and money into? The right answer is probably that you do not, at least not 100%. The post mortem analysis showed that even some experienced developers did not always have the ability to make these decisions, which consequently had a negative impact on their games. During the course of this chapter we wish to explore the subject of creating ideas and give suggestions for tools which can help gather the appropriate knowledge from which it is possible to make a weighted decision regarding these ideas. However, firstly we will look at creativity as a term in the context of design – what is creativity and how do we control it?

8.2 Getting the great idea

Being creative on command is extremely hard and you risk ending up producing little if anything at all. Every single person has his or her way of sparking the creative process; some are natural idea generators and produce one idea after another, while other needs time to think about what and how to do (Fullerton, et al. 2004, p. 9-10). Inspiration is often the key to creativity and reveals itself in the most unexpected places and ways - a walk in the woods, a memory from childhood, a billboard, etc. It is hard to pinpoint precisely what creativity is and how it occurs, due to the nature of its individuality (Rollings & Adams, 2003, p. 29-31).

We talk of innovation if creativity at some point results in a product which brings economical growth. Innovation is often confused with creativity. However, to optimize a creative process it is important to distinguish between them. Innovation can be seen as the product of creativity, meaning when initiating a creative process it might end up creating something innovative. Furthermore the target group for each is somewhat different. To be creative is to make something unique for you, while being innovative is to create something useful for a recipient, e.g. a company. This is, as mentioned, closely connected to the fact that innovation is aimed towards economic growth, while creativity is not necessarily so (Darsø, 2005, p. 158-159). But what does this mean? In order to get "the great idea" game designers must initiate an active creative process where they put their mind in a stage of awareness without limitations in terms of e.g. profit (Gold, 2004, p. 13).

Thinking in terms of profit when exploring new ideas can act as a limitation. Be creative but avoid the desire for innovation since you risk not exploring sides of a problem which at first glimpse might seem boring. Furthermore one has to acknowledge the fact that innovation in computer games has reached a point where we seldom see *true* innovation. The reason for this is like other field. New discoveries brings much attention given that it allows us understand something we never understood before. But as time passes and we learn more and more of the topic, the gap between innovation and uniformity becomes smaller. For example, at the beginning of computer games it was possible to invent whole new genres by making a game, whereas this rarely happens now. We still see innovation in games, but in much smaller scale - we refine features rather than inventing them. So the urge for innovation can be a strong incitement for doing games but it must not be the primary reason since damaging the creative process might be the result instead.

8.2.1 The process of creativity

To be creative can be explained as a divergent thinking process where designers explore different solutions. At some point in the process the focus is changed to convergent thinking where logic helps to narrow the idea down. Löwgren and Stolterman (1998, p. 57) describe a design process as a trinity consisting of a vision, an operable and a specification (see figure 11). Even though the three parts is illustrated as a trinity which affects the others the process itself is sequential - a designer forms a vision that leads to an operable plan which is followed by the specification. The vision is best described as the initial idea and can take many forms. The operable plan is the first step towards a clarification of the vision. It can be in the form of sketches, drawings, metaphors, etc. It should function as the connection between the vision and the design situation. The specification is the last step and acts as a presentation of details. The idea behind the model is that through iteration of the operable plan and the specification designers develop the vision and by doing so solve the problem area.

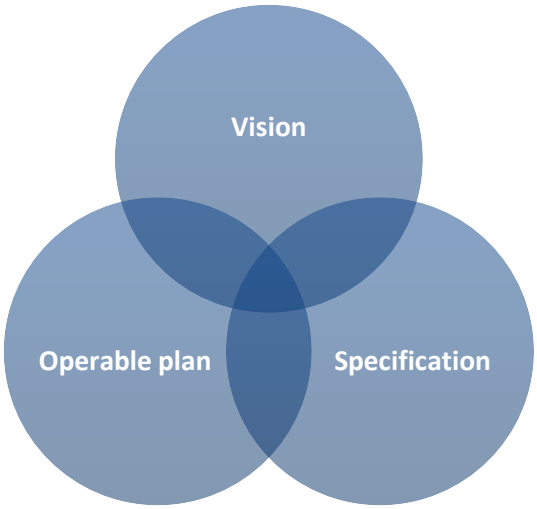

Figure 11: The design process (Löwgren & Stolterman, 1998).

Another way to approach design is through the process of creativity, described by innovation expert Lotte Darsø where a person goes through five sequential stages to formulate an idea (2005, p. 166):

1) First insight
2) Saturation
3) Incubation
4) Illumination
5) Verification

Even though this is more a mental activity it shares some similarities with the model described by Löwgren and Stolterman. Both take the premise that being creative is an active process where the visualizer has to step into a certain state of mind to produce ideas. Furthermore both suggest a sequential process to obtain the knowledge needed to make decisions concerning the value of an idea. However, the two models act on different abstraction levels. The process suggested by Darsø does not give any concrete applicability but is merely a generic way to approach creativity, whereas Löwgren and Stolterman are directly aimed towards the creation of information technology. Since Löwgren and Stolterman has a very concrete focus of their model it also entails a specific behavior in form of operations and tools, and for this reason we will use it simply as an example of how to structure such a process. We

Case study: Importance of a clear vision

The developers of *Soldier of Fortune* (2000) lacked a clear vision resulting in fundamental changes of the game during the development:

"The single most damaging problem during SoF's early development was that the original game lacked a truly focused design. We knew what the fundamentals of the game would be, but we did not have the specifics that we needed to create a solid, cohesive product. The game's overall story changed five times before it was finalized - at one point we had even changed the basic game concept to a team-based tactical shooter, similar to Rainbow Six" (Biessman & Johnson, 2000).

will during the last part of this chapter try to explain a game design phase through the process of creativity, but extend it by giving a more tangible approach in regards to game development in form of tools, as Löwgren and Stolterman does with their model. These tools should help to elaborate and verify a game idea and by doing so help to explore and develop it further.

8.2.2 First insight

Ever wonder why leaves are green? Why we drive on the right side of the road? Why gravity exists? Curiosity is the key element of the fist insight and is the foundation for getting an idea (Darsø, 2005, p. 166). By asking questions, you observe the surroundings and by doing so you start to formulate ideas of how to make things better or use them in another context. It is impossible to say what inspires people or when - it is very individual. The best way to start the creative phase of idea making is to use different sources of inspiration, whether or not it is a book, running a marathon, or taking a nap. The important part is to pay attention to the surroundings and how they affect our thoughts (Fullerton, et al. 2004, p. 140).

As mentioned the great idea can appear out of nowhere so be prepared for it. Bring a notepad, a PDA or something else where it is possible to quickly write any idea down that might pop up unexpected. To organizing the ideas into some sort of structure e.g. a database, will help recovering old ideas and at the same time use it as a source of inspiration for future projects. By doing this two things can be gained - firstly ideas which have been written down tend to be easier to remember and secondly it is a chance to dismiss all the bad ideas, but do this with care, since bad ideas in one context can be brilliant in another (Fullerton, et al. 2004, p. 140-142).

8.2.3 Saturation

Logical thinking is the key element at this stage. The stage can be described as a process of formulating and gathering data and information concerning the problem – to broaden ones knowledge base (Darsø, 2005, p. 166). The idea of knowledge gathering is a concept which has become an important focus area for many companies. No organization can afford to depend solely on the abilities of the individual - "Today the knowledge of one person is not enough" (Darsø, 2005, p. 32). At one time it only took a single person to develop a computer game, but nowadays the industry has become dependent on group structures. This only enhances the process of creativity since the opportunity to find the knowledge needed within the organization itself has become greater.

The hard task of the visualizer is to convey his idea to his co-workers so that they can give him the information he seeks. It is not easy to communicate a vision which only exists in one's head, and this all boils down to communication. Verbal communication is preferable here since we are still dealing with a vague idea and only trying to gather information to support the curiosity. This is also described by Craig Larman, who refers to The Agile Manifesto[24]:

> The most efficient and effective method of conveying information to and within a development team is face-to-face conversation (Larman, 2004, p. 28).

To describe the idea on paper is still too early and little will be gained compared to the time lost doing it. However it can help to communicate an idea if something is written down.

[24] Developed by the Agile Alliance with the purpose to support individuals and organizations that use agile approaches to develop software (www.agilealliance.org).

A concept which worked well in our experiments was the use of keywords. When you want to formulate the idea a bit more, try writing a few keywords which describes the core feature(s). By doing so you help yourself to understand it better and at the same time it gives a more tangible element which can, along with an explanation, be enough for others to get a better understanding of the vision. Try limiting the keywords to the core of the idea in this phase.

8.2.4 Incubation

The incubation stage equals time to reflect and cultivate an idea - mainly a process of holistic thinking (Darsø, 2005, p. 166). The goal is to understand the context in which the idea must function - the bigger picture. So far the process of creativity has been somewhat informal and unstructured, however for the incubation phase we suggest brainstorming as a tool to continue the creative process, which has mainly taken place inside the head up until now. There are many types of brainstorming, some requires more experience than others, but generally they all encourage the process to change from an individual to a group oriented process. The focus of a brainstorming session is to tap into people's creative mind to create many ideas fast. Brainstorming has become a relatively well known term and many see it as a tool which does not require much from the participants. However, this could not be further from the truth. Being in a brainstorming session requires a lot of focus and determination from the participants - it is not easy to be creative in a formal way. Brainstorming is much like a hammer - anyone can use it, but for a rookie it takes many tries to hammer the nail whereas an expert does it in one blow. We need to exercise our brain into being creative on command (Fullerton, et al., 2004, p. 142) and even then it can be hard to brainstorm since we rely on other people to be creative too. A brainstorming phase can be described in

Case study: Continuous idea generation

At Insomnia Games anyone can act as a game designer by sharing new ideas and thoughts with the development team:

"Everyone in the company has always been free to contribute creatively to the projects. It's not a requirement, but for those who are interested it's an opportunity to affect the direction our games take. Programmers are encouraged to contribute to story, artists are asked for ideas on design, and so on. During Ratchet & Clank, a large percentage of the team contributed ideas outside of their particular areas of expertise, making the game one of the deepest and most varied titles we've developed.
This does not imply that we design by consensus. There's a solid structure in place to ensure that we adhere to the macro design and remain consistent with the game's "flavor." But adopting an approach that encourages design participation gives us a real wealth of creativity from which to draw while enhancing the sense of ownership everyone feels in our games" (Price, 2003).

three steps; to gather a group of people, to generate ideas without criticisms or analysis and to systematize the result with the purpose of making it usable for further developing (Löwgren & Stolterman, 1998, p. 111)[25].

So the first thing to do is to gather a group of people to brainstorm with. For practical reason it may seem natural to include people who are going to utilize the result in a game developing context meaning designers, producers, artists etc. However, bringing other people along who can contribute with other perspectives may also be a good idea. The size of the group may also vary, but try to keep it no bigger than seven with three as a minimum, since this is where the best result are achieved (Löwgren & Stolterman, 1998, p. 111).

Before starting the actual brainstorming phase it is important to set some ground rules for the session. First of all put all negative thoughts and criticism aside - the purpose is to create ideas and not to analyze them. For now all ideas are welcome. Secondly, it is important that no one holds back, every little thought can be helpful and might be the piece that solves the puzzle. Furthermore the purpose of the group is to inspire each other, so by using

[25] Our translation from of the original Swedish text.

ideas made by others for improvements or whole new ideas is allowed and should be embraced (Löwgren & Stolterman, 1998, p. 111).

When participating in a brainstorming session there are multiple ways to spark the creative mind. There is no silver bullet which works for everyone, so it is up to the individual or group to find the way which fits the best. A technique which might help get started is a simple use of *idea cards*. Take a bunch of small notes and write random ideas on each, put them in a bowl and draw 2-4 random cards, whereafter ideas have to be created based on these (Fullerton, et al., 2004, p. 143). We used this technique in our experiments as a base for developing our ideas. We had around 40 idea cards which we used to spark the brainstorming phase. Combinations like "Resistance", "Smash" and "Collection" became the topic for one brainstorming session. Whether or not the final idea ends up using these words is not the point – the point is to kick-start the creative process. More experienced brainstormers might find it more interesting to use the technique called *Method 635* described by Löwgren and Stolterman. The basis for this is to have six persons who are well acquainted with the topic of the session, whereafter the task of each participant is to write three basic ideas. Afterwards these are passed on to the next participant who develops it further by adding three new ideas or modification. This is done until all ideas have been explored by all participants. Take 6 participants, which make 3 ideas, in each of 5 rounds, hence 635 (Löwgren & Stolterman, 1998, p. 113).

At some point the brain cannot handle any more creative thinking and it is time to stop - during our experiments we learned that our limit is around 30 minutes and if the sessions were to continue a break would be needed at this point. Do not force people to go on, but instead try to structure the ideas already generated. Organize the ideas into categories - are any alike, do some share similarities and so on. Do not skip or remove any ideas yet - this is the

job of the next phase of creativity. At this point you can even start to organize ideas into an early version of a design document. A collection of structured ideas should be the result of the time spent with brainstorming (Löwgren & Stolterman, 1998, p. 112). To save the work done is essential, since there might be a nugget hidden somewhere.

Again it is important to stress that formal brainstorming is not easy and takes much practice, so if the great idea occurred the first time, do it again. As the brain is trained to think creatively it will become easier.

8.2.5 Illumination

The *Aha-moment* (Darsø, 2005, p. 166) defines when the illumination stage is reached. It is described as a sudden strike of lighting which makes the solution appear clear. It tends to happen after a break from the idea process - doing something else often helps to see the idea much brighter when returning to it (Darsø, 2005, p. 166). Doing inspiring things might help to force this moment to occur but at some point it might be more practical to take a different approach. For example to continue in a more rational order by evaluating the idea, find flaws and thereby improve it. Edward de Bono has suggested a method – the *Six Thinking Hats* - to approach a discussion process with the aim to optimize and utilize the intelligence, experience and information of each participant (Bono, 2000). Our suggestion is to use this method as a tool to reach the "Aha-moment" in a more formal and time economical fashion. The idea behind the method is to separate a meeting in smaller phases where participants address a topic from a different angle in each phase - they put on different hats. There are 6 phases where each are represented by a colored hat, hence the six thinking hat. Edward de Bono describes them as following:

> **White hat** White is neutral and objective. The white hat is concerned with objective facts and figures
>
> **Red hat** Red suggests anger (seeing red), rage and emotions. The red hat gives the emotional view.
>
> **Black hat** Black is somber and serious. The black hat is cautious and careful. It points out the weakness in an idea.
>
> **Yellow hat** Yellow is sunny and positive. The yellow hat is optimistic and covers hope and positive thinking.
>
> **Green hat** Green is grass, vegetation and abundant, fertile growth. The green hat indicates creativity and new ideas.
>
> **Blue hat** Blue is cool, and it is also the color of the sky, which is above everything else. The blue hat is concerned with control, the organization of the thinking process and the use of the other hats.
>
> (Bono, 2000, p. 13-14)

Our suggestion is to use this method to illuminate an idea and evaluate it from different angles. The method was intended to be used in groups but can also be used as a tool by individuals - the important factor is that the problem area is examined from different angles (Bono, 2000, p. 22). The idea is that everyone present at the meeting puts on the same hat and addresses the topic - it is important to notice that everyone has the same hat at any given moment. Do not grant different hats to each person, since the objective is to use the diversity of every person to solve the problems or develop the idea.

Since the description earlier is somewhat shallow we wish to clarify how each hat works and the possible pitfalls concerning the hats.

The *white hat* is all about collecting information available among the participants. De Bono compares it to a computer - "We expect a computer to show us the facts and figures on demand. We do not expect computer to argue with us and to use its fact and figures only in support of its argument." (2000, p. 27). So when putting on the white hat you have to think rational, neutral and without emotional and by doing so try to uncover as much information as

possible. It is important that you do not engage in debates regarding information. If some information contradicts others there is no point in arguing about this - the information is just places parallel to each other, unless it is of such importance that it needs to be solved immediately to continue the session (Bono, 2000, p. 25).

The *red hat* is the opposite of the white - here it is about emotions, intuition and feelings. They form the basis for values and help us understand the context we are acting within, and by simply ignoring them you act against the very nature of humans. Emotions are of course not always right, but still form the basis for many decisions - we follow our intuition from time to time and based upon this we learn how to act if a similar situation should occur. Therefore a participant must always comment on a situation or idea and should not be allowed to pass, since it is a matter of personal belief and feelings - not facts or knowledge.

Many see the *black hat* as the easiest one to use - black stands for caution and help us to be careful with things that are illegal, dangerous, unprofitable, etc. (Bono, 2000, p. 73). We have a tradition in western cultures to think critical and to argue about things which are contradictory and inconsistent. We live by certain norms, values, ethics and policy and if someone is doing something against these we put on the black hat in order to address this (Bono, 2000, p. 73). The arguments of the black hat is often confused with the red hat and therefore it is important that you form the foundation of your arguments in facts and by doing so put away all feelings towards the topic.

The *yellow hat* is about optimism; how can we make things happen? It is the opposite of the black hat and may for many people be one of the harder hats to wear. Humans have a natural mechanism which takes care of the black hat; however we do not have this for the yellow hat (Bono, 2000, p.91). Using the yellow hat can end up creating a huge value for the project since we have to

look at things we would normally dismiss before taking a closer look. This however is what the purpose of the yellow hat is all about - seeing things that are not normally obvious (Bono, 2000, p. 92).

The *green hat* focuses on creativity and how we develop ideas. When using the green hat it becomes the job for every person present at the session to be creative and not only the idea-person. If you do not have any thing creative to add, you simply have to sit quiet in your chair and do nothing (Bono, 2000, p. 115). The green hat also lets you decide a course of action and to solve problems introduced under the black hat (Bono, 2000, p. 116). Creativity is the purpose which hopefully should lead to improvements of some sort and create a better foundation for the product.

The *blue hat* is concerned with the process of thinking - thinking about thinking. In this phase you decide the agenda for the session and how you wish to explore it. Everyone is part of this phase to begin with, but as the hats change for everyone else one person, the facilitator, remains in the blue hat phase and directs the rest during the other phases. His job is to ask for the outcome of a phase, e.g. a conclusion, a solution, a decision (Bono, 2000, p. 147-148).

The method of the six thinking hats is a way to help explore a subject more thorough and by doing so reaching the design phase with a clear idea of where and what you are doing. At first glimpse the method might seem very similar to the brainstorming methods mentioned earlier, but whereas these tried to create ideas, the six hats method is about developing an idea into something more tangible. You are being more than just creative - the method expects the designer to look at the idea from more angles by breaking the thinking process into small manageable sizes. Brainstorming is all about quantity where the six thinking hats are about quality.

During the course of our experiments we used the six thinking hats as a tool. Early in the creative process we noticed that we did not use much time exploring our ideas to the fullest. We experienced misunderstandings from time to time, since people had different opinions and perceptions regarding the ideas which had to be taken into account. By using the hats we were able to approach it in a more structured way were everyone had the chance and time to comment and explain their thoughts without criticism and interruptions. By using it we were able to create more elaborated ideas for game concepts.

8.2.6 Verification

Following the illumination phase it is time to translate the insight into a more concrete solution in order to evaluate and verify it. Lotte Darsø explains it as a phase where "the idea or solution is 'rationalized' through processes of logic" (2001, p. 166). We are concerned with creating games and since this is a medium which is hard to verify without playing the game or parts of it, we suggest a more tactile approach to test an idea, in the form of prototyping. It is hard to test something which only exists as words, so by translating these into a prototype the designer will be able to verify it and determine whether or not it actually is fun.

The concept of prototyping is used throughout many industries and companies as a mean of developing ideas, however this versatility have also created many definition of what a prototype is (Hunt & Thomas, 1999, p. 53) . In software engineering it is defined as a way of "[...] giving the user a system which is incomplete and then modifying and augmenting it as the user requirements become clear." (Sommerville, 2001, p. 174). In game design, prototyping is "[...] to create a working version of a formal system that, while playable, includes only a rough approximation of the artwork, sound and fea-

tures." (Fullerton, et al., 2004, p. 157). Both definitions talks of a system - an assemblage or combination of things or parts forming a complex or unitary whole[26] – however we wish to break prototyping down to even smaller parts. By focusing on a whole system it requires a lot of resources and implementation to reach a point where the idea can be verified. These system-prototypes are useful but serve another purpose which is not important at this point in the process. Instead we will use the concept of *lightweight prototypes*. These have the purpose of exploring a limited area of a game - more specific a *game mechanic*- and should work as a simple and cheap way to quickly evaluate an idea or parts of it. It is important to stress that these prototypes should not be seen as a game, but merely a very small part of it. System-prototypes are first playables, vertical slices, alpha versions, etc., while lightweight prototypes are idea confirmation tests. A lightweight prototype could be a test to see how a character should jump, a simple interface structure, a path finding algorithm, and so on. You do not need the whole game to test how a character should jump – it is a matter of abstraction level. Seeing a simple dot jumping from one platform to another will do just fine during this phase, in contrast of waiting for models, sound, environments etc. to be done. Almost every part of a game can be broken down to manageable pieces, which can be tested in the form of a lightweight prototype. If 20 people and six months are needed to complete a lightweight prototype the scope is too large. These prototypes should quickly be done and take little resources to complete – it is a prototype not a final game. What we want to achieve by making these prototypes is to translate the insight into knowledge.

[26] Definition of 'system' found on www.dictionary.com

An aspect which is important when doing lightweight prototypes is the acceptance of failure. No one can get it right every time and as Woody Allen once said "If you're not failing every now and again, it's a sign you're not doing anything very innovative". However trivial it may sound, the important thing is to learn from the mistakes made. The fact that things go wrong and that some ideas go from being fun when talking about them to being miserable when implemented supports the idea of making small and cheap lightweight prototypes. If a game idea ends up being scrapped you can take comfort in the fact that developing costs were kept to a minimum. This could have been fatal to realize 12 months into a multi-million dollar project. To accept failure is one thing, but we could even go as far as to say that failure should be embraced. The fact that designers fail means that they are pushing the limit to the maximum and probably crossing it too – this is the essence of making lightweight prototypes; to test things which would require too much time and money later on in the process. Earlier we talked about innovation and how designers should not restrain themselves by thinking in terms of innovation. However, this is the chance to create something new. See the possi-

Case study: Playable prototypes

By using prototyping when developing *Deus Ex* (2000) it was possible to test and play the most crucial parts of the game quickly – even though it was hacked together it served as an important way to understand the game better:

"One example of where our proto-mission idea was successful was in May 1998, when our milestone was to have prototypes of critical game systems in place and two test maps running, in this case the White House and part of Hong Kong. The maps were crude, the conversations raw, and the game systems hacked, but we could see -- and show -- the potential. To our advantage, we resisted the temptation to do just the stuff we knew would work and the stuff that would look the prettiest, and prototyped new, risky stuff first. Conversation, interface, inventory, skills, and augmentations were all at least hacked in so we could see them in action. The White House was likely to prove our toughest map challenge, so we built it first. (Almost unbelievably, I missed what may have been the riskiest, most critical game system in all of our early prototyping, NPC AI. I should have insisted on early prototyping of our AI but I didn't.) With the proto-mission system, we could immediately see some of the limitations of our technology" (Spector, 2000).

bilities in prototyping and create as many solutions as imaginable. If we push the creativity to the limit we might end up with something new and exciting.

8.2.7 Game mechanics

In relation to lightweight-prototypes we wish to specify what we mean by game mechanics, but in order to do so we must first take a closer look at games and its components. Andrew Rollings and Ernest Adams (Rollings & Adams, 2004, p. 9) talks about games as a synergy between three parts:

- Core Mechanics
- Storytelling and Narrative
- Interactivity

Where each part, in interplay with the others create the experience a player attains when playing. Core mechanics of a computer game can be compared to the rules of a board game. They form the basis for the game and define the operations available within the game world – the foundation of gameplay. "Defining the core mechanics is the "science" part of game design" (Rollings & Adams, 2004, p. 9) and if you are not able to do this you risk ending up with a poor game. Furthermore, it is important not to disguise technologies[27] as core mechanics since these should not be important to make the fundamental of the game work. The problem with technologies is marketing. Some games are based and marketed with technologies as USPs in order get the attention of the customer which again helps to sell the product. With a limited budget and time frame the money available will often be spent on the parts which sell the product the best (Rollings & Adams, 2004, p. 9-10).

[27] By technologies is meant elements which can be seen as a technological achievement in relation to what there is currently available on the market, e.g. improved graphics, realistic AI or authentic voice acting.

Rollings and Adams separate storytelling from narratives in the sense that every game has a story, but not all games have narratives. The story of a game is created as the player plays the game – it emerges from the interaction with the game. On the other side narratives, as they use the term, is when part of the story is being told to the player by the designer. It can be compared to the story of a book – you read a book but cannot interact with the story or change the content (Rollings & Adams, 2004, p. 10-11).

The last component is interactivity and handles everything the player sees, hears and acts within the game's world. Graphics, sounds, interfaces are all examples of elements which is part of interactivity. "Poor interactive design ruins many products" (Rollings & Adams, 2004, p. 12) – we probably all experienced a game where the interface or control scheme has been so badly designed and by so shattered the whole play experience even though the game had some potential. Especially graphics has with the increase in computation power become an even bigger part of game design. However it is important to stress that a good looking game does not make up for badly designed mechanics. The hard part is to weight the two against each other in matters of time, manpower and money.

The definition presented by Rollings and Adams has certain qualities which we want to adopt in relation to lightweight prototyping. The concept of core mechanics fits our way of thinking in regards to verifying game ideas through prototypes. We believe – as Rollings and Adams do – that these are the essential part of a game design process and must be the main focus of the design process. However we wish to extend the concept by removing elements such as controls and interfaces from the term interactivity and add it to the core mechanics. We wish to distinguish the essence of a game from the wrapping. Graphics, sounds and story is wrapping – even though these can be an important part of a game it is not the focus for the exploration of game

ideas, and consequently our approach to lightweight prototyping. To write a story or to make a character is not the job of a game designer. It requires some other tools and is not important in the context of lightweight prototypes.

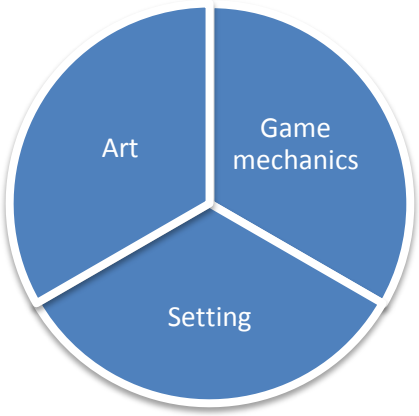

Figure 12: The tripartition of a game.

However since elements like interfaces and controls play a vital part in how the core mechanics work, we have chosen to incorporate these along with core mechanics into a common term called game mechanics. We have also chosen to rename interactivity to art since the elements which make it interactive have been removed and now only contain visuals and audios. Furthermore Rollings and Adams chose to distinguish the term story from narratives however this separation can create misunderstandings. The blend between when the game is presenting a story and the player creates a story is not black and white. For example when a player is presented with options in a cut scene, does the story become a narrative or story? We see many shades of grey in this separation and have therefore chosen to discard this term. We use the term *setting* instead, which contains elements such as background story, theme, universe, scenario, setting, etc. The important thing here is that we have eliminated the player interaction with the game as a key component. What we are trying to define is how a game designer can look at a game to

understand the parts he needs to design – we are not defining the concept *game*. We also recognize that not every part of a game is explained by the divisions themselves. However they can be explained as the sum of two or all parts – e.g. a level can be explained by taking elements from game mechanics, setting and art. Based on this we define the key components of a game with regard to game development as a trinity of game mechanics, setting and art.

8.3 From theory to practice

As mentioned earlier the creative thinking process is a description of how humans generate and formulate ideas; however we have tried to use it more concretely in the context of game development. The process is divided into 5 phases – (1) first insight, (2) saturation, (3) incubation, (4) illumination and (5) verification. What we have tried to do is to introduce a set of tools which can help to elaborate ideas; keywords for communicating, brainstorming for generating, the six thinking hats for formalizing and prototyping for exploring and validating of ideas. Even though the process may sound trivial it is harder said than done. The tools introduced are all about "state of mind" and even experienced creative people can lose their spark once in a while. During our experiments it quickly became clear that this is something you need to practice on. Failure will occur many times at the start, but with practice the brain will get used to thinking creatively whereafter the stream of ideas will happen more frequently.

9 Feedback and communication

Feedback is a tricky business, and the art of communication is not that easy to master either. In this chapter we will take a look the values of feedback and communication, and see how a team can implement feedback loops and the rapid feedback into their productions. A feedback loop is when feedback is used to generate new feedback through a process of continuous improvement, and it is the very essence of rapid feedback. We will also look at some tools which can help communicating the feedback in useful ways.

On May 27, 1968, the US Navy submarine *USS Scorpion* went missing on a routine mission in the North Atlantic while inbound for Norfolk, Virginia. The US Navy used every waking hour and many different methods in trying to find it, but none bore fruit. They had to face the fact that they had lost one of their most lethal and valuable weapons, a nuclear submarine. When all options seemed only to lead to dead ends the Navy brought in naval officer John Craven. His job was to recoup what was left of the salvage operation and file a report about it. A report that undoubtedly would be stored in a cabinet never to be opened again. That the US Navy could lose track of a submarine was unheard of. It was an embarrassment that no one seemed eager to talk about. However, Craven had other ideas. He put together a group of experts from a wide range of fields, including mathematicians, submarine advisors and salvage professionals. Their job was to come up what they believe would be the best guess of where the submarine sunk based on their respective background. They made many different scenarios with different factors included. They each made scenarios of where the submarine was heading and could have happened to it to make it sink. Craven then took all the scenarios and calculated an average location by using Bayes' theorem of probabili-

ty[28]. He used this newly calculated average to start up a new search mission. He was determined to find the submarine and he was sure that this new location was the right one. And he and his team were successful. In fact, they were so successful that they found *USS Scorpion* lying on the bottom of the ocean less than 200 meters of where the average of the Bayesian model said it would be. John Craven had done what the US Navy was not able to. By combining the knowledge and experience of *many* experts, he found what no individually lead salvage mission was able to. (Sontag, Drew, & Drew, 2000, p. 88-123)

This story illustrated the power of collective knowledge. By getting input from people with different backgrounds the feedback will be more complete. Allowing, for instance, graphical artists to follow discussions among programmers can help the kind of "outside the box" thinking creative processes like game design require. Having open and free access to information about the project is a good way of facilitating this kind of cross-disciplinary knowledge sharing. If the information is restricted or difficult to access, getting feedback from the different sections of the development becomes harder and the feedback itself less valuable. Sharing and enabling easy access to relevant information is the first step towards tapping into the full potential of your team.

By creating a sense of ownership among the individual team members, you run the risk of making everyone a designer. There should be a small team of designers who are responsible for the design of the game, but that does not mean the ideas from the rest of the team should not be taken into account. The designer team is the critical filter for ideas and feedback from the rest of

[28] In short, *Bayes' Theorem of probability* is a formula that is used to compute posterior probabilities by revising prior probabilities. Read more on Wikipedia.org (http:/ / en.wikipedia.org/ wiki/ Bayesian_probability).

the development team, maintaining the same overall look as John Craven in his search for the lost submarine. A common vocabulary that makes it easier to discuss ideas and give feedback to each other, and this also promotes a sense of shared responsibility and ownership. Another way of easing communication between the different team members is to find canonistic gameplay elements and ensuring that all team members have the same interpretation of these. If everyone uses the same words for describing their feelings and opinions about mechanics or design choices, the process of giving and receiving feedback will run much smoother and with less misunderstandings.

9.1 Short feedback cycles

For all product development processes, one of the most important elements is the feedback cycles. Designing in a vacuum is rarely a successful approach and has often lead to products that do not live up to user demands and expectations. The longer the feedback cycles are the longer it takes to spot mistakes and problems with the product and the longer it takes to fix them. The aim is to have as short cycles as possible. The principles presented in this chapter all have the overall goal of *compressing* the feedback cycle in order to make the distance between the initial idea and a playable version of the game much shorter. Currently it can in some cases be as long as one whole year (Biessman & Johnson, 2000), or at least many months. If it was shorter the feedback earned from play testing would be available earlier when there was still time to correct the errors.

Larman underlines the importance of rapid feedback by paraphrasing a UK study on 1.027 software development projects where 82% of the projects report that using the waterfall method or similar sequential development methods is the number one issues that lead to the overall failure of the project:

> This suggests that [...] the approach of full requirements definition followed by a long gap before those requirements are delivered is no longer appropriate. The high ranking of changing business requirements suggests that any assumption that there will be little significant change to requirements once they have been documented is fundamentally flawed, and that spending significant time and effort defining them to the maximum is inappropriate (2004, p. 74).

Rapid and frequent feedback reduces project uncertainty. It allows the developers to make frequent adjustments and to learn from them (chromatic, 2003, p. 8). Defective elements are quickly discovered so that new actions and directions can be taken. When reading the postmortem articles is becomes apparent that the view of most developers is, that in order to test the "fun factor" properly the game has to be in a state of publishable quality. It is not uncommon that games go through a very long process before being tested for playability. Even *The Cerny Method* proposes a rather long feedback cycle for testing playability. They speak of a *first playable* as the testable version of the game. However, we can see from the problems mentioned in chapter 4 and 5 that there is a need for faster and earlier feedback.

The Cerny Method has proven to be a very useful and successful method of developing games. But it is missing the early feedback many of the agile software development methods are proposing. The gameplay can be tested

Case study: Short feedback loops

Late feedback during the production of *Trade Empires* (2001) resulted in a wrong order of priority:

"If we had done more beta testing, with a larger group and earlier on, we would have gotten the kind of outside feedback that would have helped us realize that some of the tradeoffs we were making were going the wrong way. We didn't miss some features nearly as much as other people did -- we were so used to the game that we adjusted our play style and underestimated how important the missing features were.
We did our best to show the game to as many veterans and rookies as we could get into our office. But just not enough eyes saw Trade Empires, especially without having a Frog City person at their elbow to explain away any possibly ambiguous elements of the interface or game rules" (Bernstein, 2002).

much faster by doing *lightweight prototyping*. Creating small prototypes of insulated gameplay mechanics can be a very fast way of determining if the design is properly defined. There are many tools available on the market for prototyping. We will not go as far as to recommend a specific one[29], as it often comes down to personal preference and level of experience. Developers looking into this should use the tool they are most comfortable with and which fits the game they are making.

9.2 Collecting feedback

The knowledge gained from the design process must be conveyed effortlessly to other team members. One approach is to use a Wiki system. In short, a Wiki system is a website where the users can alter and contribute to the content of the site by adding and revising the stored information. If a user deliberately or unintentionally changes or erases part of the content, it can be corrected very easy by using the built-in revision system that tracks all changes and the original information can be retrieved. The most famous use of a wiki system might be Wikipedia.org, a free user-driven encyclopaedia written and edited by experts, non-experts and enthusiasts from all over the world. It is formidable way of relinquishing control over the voices of design. It is exactly this form of non-management that has paved the way for huge success of the internet (Weinberger, 2002, p. 23). It is about cutting out the middlemen and letting the inmates run the asylum.

[29] There are numerous tools available on the market for lightweight prototyping, both free and commercial ones. To name a few; Microsoft's XNA Game Studio Express (free – very advanced IDE), Google's SketchUp (free – flexible 3D drawing program), GameMaker (free/ commercial – flexible 2D game maker), Blitz Basic / Blitz Max (commercial – flexible 2D/ 3D IDE), Adobe Flash (commercial – advanced vector based 2D drawing and programming IDE), PyGame (free – Python programming framework), Blender (free open source – flexible 3D drawing / programming environment).

9.2.1 Sharing

Tim Ryan (1999) proposes some guidelines where every phase of the game development process is covered from idea to release. The aim is to create a document that describes in detail how to handle every situation before the project starts. Based on the document the production is a matter of following the directions described, until every feature is implemented and the software is ready to be tested and released. Ryan's guidelines for creating the needed documentation fit with this belief of an all-knowing designer. Ryan's thorough description of how to write a game design document can be useful for storing the knowledge of a project but it is problematic to believe that it is possible to predict all problems before the production starts. Object oriented analysis and design theory (Mathiassen et al., 2001, p. 15) emphasize that the benefits of a design document is to create a connection between the different stages in development. The document should be brief and accurate to allow focus on the important parts. The reason for this is to stimulate the creative effort within the development team by harnessing inspiration from the design document.

The agile method Scrum use daily 15 to 20 minutes stand-up meetings to facilitate knowledge sharing. The purpose of the meeting is to update the team on the progress of each team member, and if a problem occurs it is the team's responsibility to solve the problem together (Schwaber & Beedle, 2002, p. 40-47). If the company using this practice is afraid to fail and sees this as a competence problem with the employees, the stand-up meeting can be a problematic and an unpleasant experience. If team members feel intimidated by these meetings, the purpose is lost. In this case, the openness that should be the result of this approach will be nonexistent. The organization needs to support the selected method 100 percent, and embrace a free communication

structure that can illuminate problems, and at a later stage turn them into challenges that can benefit the production

9.2.2 Short and simple

> *The Landing Pilot is the Non-Handling Pilot until the "decision altitude" call, when the Handling Non-Landing Pilot hands the handling to the Non-Handling Landing Pilot, unless the latter calls "go-around," in which case the Handling Non-Landing Pilot continues handling and the Non-Handling Landing Pilot continues non-handling until the next call of "land" or "go-around" as appropriate. In view of the recent confusions over these rules, it was deemed necessary to restate them clearly.* ***- British Airways memorandum, quoted in Pilot Magazine, December 1996***[30]

It is important to maximize the information that is passed to others in order to prevent any information getting lost in the process. The Lean method recommends using an A3 sheet of paper as the standard documentation form for projects[31]. This forces the designer to rethink the level of complexity of the information he is trying to communicate (Poppendieck & Poppendieck, 2003, p. 157-159). A single sheet of A3 paper might seems as a very small area in which you have to fit large amount data into, but this is exactly why this exact size is useful. If the data is any larger it should be split into more documents instead. If using a Wiki or similar system, artificial restrictions can be inserted into the system that forces the writer to uses less than e.g. 5.000 characters. Large documentation could confuse the readers and might leave more questions than they answer. Often the reader will seek out the original

[30] This quote is reprinted as written in Andrew Hunt and David Thomas' *The Pragmatic Programmer* (2000, p. 217)

[31] Poppendieck and Poppendieck lists in their book *Implementing Lean Software Development* (2007, p. 158) a very useful, although short, checklist for creating an A3 documentation. The checklist contains advice such as use as few words as possible, and encourages the writer to use figures and graphs to underline the information.

author in order to get verification on uncertainties that might have arisen from reading the material.

Another thing to be aware of when sharing knowledge is that tacit knowledge is hard to communicate in a document, and up to 50 percent of the information may be lost when information is passed on the others in written form. After two handoffs there is only 25 percept of the original knowledge left, after three handoff it is down to only 12 percept (Poppendieck & Poppendieck, 2007, p. 77). Lean suggests that the best alternative is to use face to face communication to answer all of the questions that may arise in a handoff situation. Face to face communication is useful on a daily basis and can help to solve urgent problems. The aim is to have as few handoffs as possible and that is where a Wiki system or another system with easy access is applicable. When information is shared it is important that the content is not diluted in the process. The tacit knowledge the original author processes is not always easily conveyed in written form. The A3 document can be a good

Case study: Feedback enhances the game design

The designers of *Deus Ex* (2000) learned the hard way that good ideas on paper might not be so good in reality. Instead the only way to know is to play it and get feedback:

"When Gabe Newell from Valve came down and played our prototype missions, he correctly identified the utter lack of tension in our skill and augmentation use, as written up in the design doc and ably implemented by the coders. The worst was confirmed when Marc LeBlanc, Doug Church, Rob Fermier, and other friends from Looking Glass Studios and Irrational Games played the proto-missions and came to the same conclusions. Actually using skills and augmentations revealed things that merely thinking about them could never have revealed.
We took the criticism, and with it in mind, lead designer Harvey Smith revised the skill and augmentation systems pretty thoroughly, proposing an elegant system of consumable resources and time passage, all tied to skill level. This increased the tension level, provided new rewards, and allowed players to think and make informed decisions. Harvey also proposed a revision to the augmentation system, introducing an energy cost for their use (something I had foolishly rejected earlier on). Again, this gave us the opportunity to hand out items that would replenish energy -- in other words, we instantly had more things to hand out to players as rewards. It also introduced a level of tactical thinking to augmentation use that makes the system work. None of this would have happened without prototype missions and some harsh (but fair) criticism they allowed" (Spector, 2000).

choice in combination with face to face communication. Beware though of the pitfalls of oversimplification. All possible questions must be answerable by the documentation. Do not dumb down the documentation, which will only lead to similar problems as over-documentation.

9.3 Press 'OK' to cancel

Canceling projects have always been an unmentionable part of the game industry. It is something you always fear and speak of in hushed tones afraid that someone with the authority to actually cancel your project will hear you. But as Cerny and John (2002) states so clearly, canceling a bad project is good. If the feedback is suggesting that the design you are currently working on is not going to become entertaining for the player, cancel the project in time. Canceling projects is not about punishing the development team for sluggish work or not meeting deadlines. It is about saving time and money and instead focus it on new ideas and concepts. It is as the team behind the Experimental Gameplay Project at Carnegie Mellon University so poetically stated it: "Heavy Theming Will Not Salvage Bad Design (or 'You Can't Polish a Turd')" (Gabler, Gray, Kucic, & Shodhan, 2006). Cancel the project before you have spent too much time and effort on it and move on to other things. Even though development so far has taken up months of valuable pre-production time it is always better to cancel a bad project and move on to others instead of forcing yourselves to work on something that both your gut feeling and the feedback has proven to be a subpar product. "A penny saved is a penny earned" as Benjamin Franklin once so wisely stated it.

Properly implemented feedback cycles can save the development team precious time and energy by being delivered on time. The important thing for feedback is the speed and timing of it. Excellent feedback delivered too late is

less useful than mediocre feedback voiced when actions could be taken to implement the concerns expressed.

10 Flexible Design

"You could not step twice into the same river; for other waters are ever flowing on to you." - ***Heraclitus***

Without doubt the more interesting trends to come out of the analysis of the postmortem articles, was the fact that the game developers who started out with a small initial concept and gradually expanded the design over time, ended up with having a more constraint free and relaxed development period and ultimately ended up with a greatly polished product. In the light of hindsight, this trend might not be the most surprising but clearly not the one we were expecting to find. At the other end of this spectrum were the "hard core" developers that created the games that ended up winning one or more awards for their technical achievements. They were, rightfully, very proud of their games, but also admitted that it had been a long and exhausting journey, one that often involved long hours and little social life, for some even to the extent losing their significant other due to the high workload. Moreover, game development should never become so important that you lose yourself in the process. When personal expenses becomes this high the development process is deeply flawed[32]. Not that incremental development approach eli-

[32] Chris Taylor, the creator of games such as *Dungeon Siege* as mentioned earlier in this report, gave a keynote speech at the D.I.C.E. Summit 2007 (a yearly conference hosted by The Academy of Interactive Arts & Sciences) in February 2007, about the importance of not working too much or too hard. Two quotes from the speech are worth referring: "Creators don't stop creating when they leave the office" and Taylor said that at first he questioned whether or not publishers, who are investing millions of dollars in a game, would appreciate how game creation at Gas Powered would come after health and family. "They were okay with it" because they have kids too, he said. "The industry is growing up". That this comes from Chris Taylor makes it all the more interesting, since any given person on the development team on his last game *Dungeon Siege* had more than 1.500 man-hours of overtime. A staggering number by any standard. He realized that this had to change and the entire culture at his game development company Gas Powered Games was changed. On their latest game *Supreme Commander* they managed to get this number down to about 100 man-hours. The game itself is as of writing this getting rave reviews from around the globe currently having a MetaCritic rating of 90.

Case study: Early prototyping

The development of *Ratchet & Clank* (2002) involved a lot of prototyping in the early phases in order to determine the design of the game:

"Even though the concept behind Ratchet & Clank was ambitious for us (integrating RPG elements into an action-platformer), we were careful not to cram too much stuff into the initial design.

[...] For these reasons, we planned the game layout much more carefully than we had on past titles. We had a pretty good idea of how long it would take to build each level, but we also knew that plenty would go wrong during the production process. So even though we had time to do 20 levels, we cut back to 18 at the very beginning.

We also made sure that nothing went into the design unless we were very sure that it was going to work. Early prototyping was the key here, but so was an attitude of general restraint. There were a few wild concepts that everyone was excited about, but had we integrated them into the macro, the project probably would have slipped. Ultimately, we were able to put about 90 percent of what we planned into the game - a record for us" (Price, 2003).

minates the overtime or the hard work involved in making games, but it seems that the developers had more mental energy left. It is as Julian Gold states it "Start small. Get bigger through small incremental steps" (2004, p. 11) that seems to give the developers more energy and incentive to "think outside the box".

Flexible design is the very core of adaptive game development. Without flexibility in the design, none of the other parts would be achievable. It goes without saying that you have to be flexible in order to be adaptive. In this section, we will look closer at tools such as *set-based* and *modular* design. We will also look at the importance of delaying your decisions until you have more, and less incomplete, information at hand to base decisions on. There is a rather consistent myth in the software and game development industry that freezing the design is helpful for controlling production and keeping it on track.[33] According to the myth, adding changes later in the production will

[33] For example of this myth see Rollings & Adams (2003, p. 17), Fristrom (2002, p. 50) and Reinhart (2000) and many more of the postmortem articles.

delay shipment of the final product substantially and equally increase the overall cost. The idea is that it gives the developers much needed constraints to working within by freezing the design. Keeping the fundamental gameplay concepts open for redesign might for some seem as a nightmare scenario. Agreeing on specific solutions and then building the design on top of this certainly is a much more manageable approach but as we will try and emphasize with this section, it is also one of the reasons why many games struggle to remain coherent and have a well-designed of gameplay experience. Being flexible in the design is much more than just having more options for color styles for the main characters. It presents a fundamental paradigm shift in the mindset of the developers. Designers must have the courage to make bold decisions that might go against the initial ideas but seen in a broader perspective will lead to a better game. It is about not being afraid of trying out numerous ideas even if the first ones are working. The reason we feel that flexibility is so paramount in game development is that it is unrealistic to design everything up front. Almost all elements of game development will change over time and not planning for it is unwise.

The important thing for game development is to be able to cope with the changes that unavoidably will arise during development. These changes might be coming from external surroundings[34] or internally in the form of peer feedback. It is naïve to plan your development in the hope that no problems would emerge in the process. For these reasons, traditional sequential development methods are not well suited for game development, as many of the authors of the postmortem articles also clearly state. Implementing new or changed elements during sequential development is very hard and often

[34] The surroundings in this case includes, without excluding any, the market, the publisher and other financial interests.

Case study: Iteration in the design

Things on paper do not always turn out as envisioned when implemented in playable prototypes, as the developers of *Age of Mythology* (2002) learned:

"Ensembles's basic design process is to get the game playable early and then tweak it until it is fun. This applied to virtually every feature in the game. Some features changed a million times, and we were willing to abandon things that just didn't work, even when it was painful.

Age of Mythology's God Power feature is a good example of this process in action. On paper, our initial concept of God Powers and Heroes sounded good: Heroes would have a button on the interface to target God Powers wherever the selected Hero happened to be - simple enough.

Unfortunately, when we got the feature in the game and started playing with it, it was awful. Having to have a Hero in the place where you wanted God Power devolved all combat tactics to selecting all you units and clicking on the enemy Hero. This led to Heroes constantly getting killed" (Fischer & Street, 2003).

leads to delay since much has to be redesigned to cope with these new elements.

Agile software development on the other hand, is, just as the name implies, a way of developing software that is more on its toes than "traditional" software development. In order to achieve as short feedback cycles as possible game development must take on some form of iterative (or agile) development process. Doing sequential development is not a feasible process when creating software products that must live up to shifting markets and/ or changing requirements. While the vast majority of game development industry is developing their games with sequential development methods, the ones applying a more agile approach clearly state this as an advantage.

In game development you constantly have to reassess what you have and make sure that the production is still on track. Alternatively, you often have to struggle hard to get the product finished even though you are aware of shifting markets and/ or erroneous products. Iteration is, as Julian Gold says; "[...] how Mother Nature does it, after all. [...] Iteration is so fundamental that it is probably impossible to avoid it" (2004, p. 11). Instead of making a big

game all at once, it could prove to boast moral and publisher confidence if the game is developed from a very humble and limited core starting point, and when proven to work, more features could be added. This also has the added bonus of the game being playable throughout the entire production. This would eventually lead to a better and more balanced game because deficits and weak areas are spotted early on.

Retrofitting new gameplay elements into the design is seldom a productive way of working. Retrofitted elements can in most cases easily be spotted and are often singled out in game reviews as being tacked on or out of place[35]. Often great gameplay emerges from the synergy of many different game mechanics and exactly therefore all elements have to be redesigned or reconsidered in order for the game to work completely satisfactory. Designing the individual mechanics in a vacuum can also lead to a lack of overall vision leaving the game fragmented or unstructured.

10.1 Playing the game

There are basically two types of design approaches, one that ensures that the execution is done properly and one that ensures that the end product is applicable. It is about either "building the product right", or "building the right product".[36] They may seem similar both in name and execution but they foster very different development views (Sommerville, 2001, p. 420). As seen in

[35] The game *The Legend of Zelda: Twilight Princess* (2006) was in development in-house at Nintendo for many years before it was eventually released for both Nintendo's current consoles; the GameCube and the Wii. Since the game had been in development for many years it was initially planned as a GameCube title, but the release of Wii required that a high profile game such as Zelda would be released simultaneously for the new Wii console and the older GameCube. The problem however was the control scheme for the Wii is radically different from the GameCube and many reviews, namely GameSpot.com, also noted that this control schemed seems very much out of place and clearly retrofitted into the design.

[36] Note the order of the words *right* and *product*.

the postmortem articles, the game industry has historically been very concerned with building its product right, making sure that it worked properly on a technical level[37]. This might originate from the early years of the industry where it was not uncommon for computer games to be developed by a single person or very small teams. Most, if not all of the people working in the industry had a background in engineering; hence they were very concerned with reducing development risk, since game productions were becoming larger and more expensive. Traditionally this was done by outlining all possible scenarios and working out all the difficult design on paper or in large prototypes before deciding on whether or not to start up a full production. If this was decided then the game "just" had to be developed according to the design documentation.

In traditional software development the discussion about which approach is right has turned almost religious with both sides arguing that theirs is the only proper way of developing software product. In connection with computer game development there really is only one right way of doing it; *building the right product*. Even though patching might prove to be the bane of the entire industry non-engaging gameplay is an even bigger threat. Computer games are all about having fun, or at least being entertained in interesting ways. This is exactly why technical achievements in computer games matters less compared to entertainment value. Seeing technical stunning graphics helps little if the "fun factor" is absent, and the only way to cut this Gordian knot is by incorporation of rapid feedback into the production and building flexibility into the design.

[37] You could always argue against this by looking at the increasing number of patches that continues to plague any PC game (and now also internet connected consoles).

10.2 Delaying decisions

Making decisions that are based on incomplete information is illogical at best and fatal for the product at worst. Wait until the *last responsible moment* (Poppendieck & Poppendieck, 2003, p. 57) with making decisions. The important thing to note here is the "last *responsible* moment". Just as it cripples the project if decisions are made too early, the impact of decisions made too late is equally disastrous. The U.S. Marines works towards the *70 percent solution*, the decisions should not wait until the ever angle is figured out and thoroughly calculated. It is often ineffective in compared to deciding to move forward. Recklessness in decision making not only connected with making hasty decisions. So when is now *now*? When do you have enough information at hand to make a sound decision? Rule of thumb; if you can wait, then you should wait, if not, decided on the information you have (Poppendieck & Poppendieck, 2003, p. 57). That is the hard part, and most likely you will not be able to find the perfect moment, but you must always have a notion of the last responsible moment in mind. Experience from and general knowledge about the problem domain also helps to demine when to maintain options and when not to (Poppendieck & Poppendieck, 2007, p. 32).

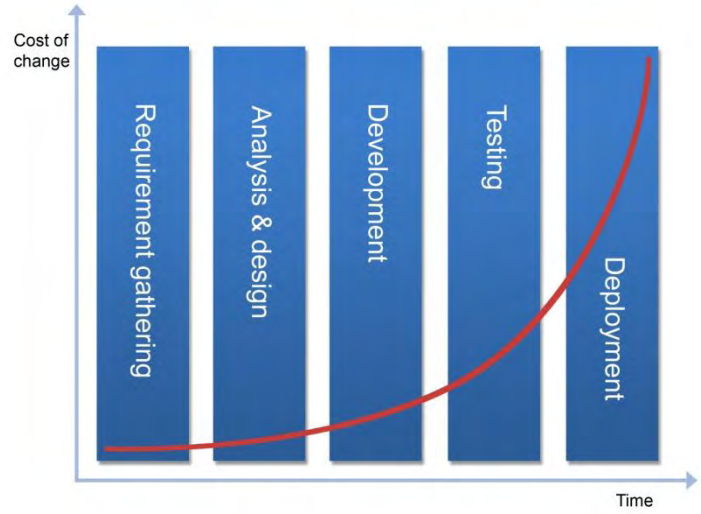

Figure 13: The assumed cost of change rises over time as the production proceeds (Beck, 2002, p. 21)

The curve in Figure 13 shows the assumed cost of change in a normal production. Changing elements later in production will cost more than elements changed early. This is most certainly true in traditional sequential developed methods such as the waterfall method. Much has to be redesigned and the project might be forced to redo the initial design phase again, leading to shipping delay and increased costs. That is exactly why the notion of delaying decisions is so important. Christensen & Kreiner (1991) speaks of the *contextual uncertainty* that surrounds the project. It concerns the dilemma of having to make decisions based on a very limited knowledge. Christensen & Kreiner writes: "[…] a project's actual results very likely are evaluated on a changed knowledge base and on other premises than they were founded on." (1991, p. 43)[38].

[38] Our translation from the original Danish text.

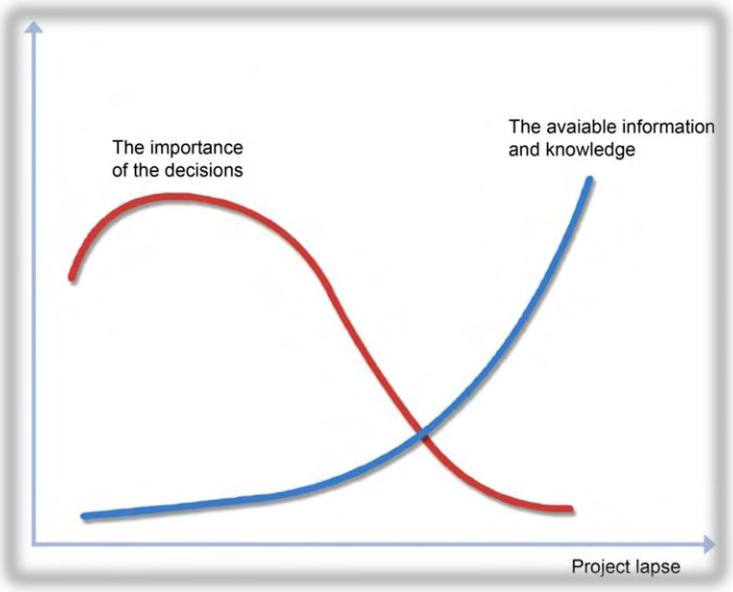

Figure 14: The *contextual uncertainty* is much higher in the initial phase of the production, when little information is available (Christensen & Kreiner, 1991, p. 41)

That is why freezing the design too early in production will lead to increased costs, because most elements in game development changes over time. The need for flexibility in game development call for another and less sequential development method in order to result in successful products. It is all about keeping your options open[39].

10.3 Set-based design

An excellent method to keep the design options open is to incorporate *set-based* design. Set-based design goes back a long way and halfway around the

[39] There are times when some elements of the game in development have to be frozen. Games in already established series usually have very limited room for character innovation and or series-defining gameplay mechanics. Games such as *Warhammer 40.000* would seem strange and out-of-character if they did not feature *The Space Marines*. For others, like game developers *Naughty Dog*, starting all over has worked very successful with their *Jak & Daxter* series. The second and third installment in the series was a far cry from the original game than founded the series.

globe. It origins from Japan and to get a bearing of what exactly that it is, we need to go back to where it all began.

10.3.1 Land of the Rising Sun

In 1921, Kiichiro Toyoda joined his father's thriving loom production company Toyoda Automatic Loom Works. The future was looking bright and prosperous for the Toyoda family business[40]. In fact it was going so well that Kiichiro wanted to expand the production to include automobiles. He went to the only place in the world for hands-on knowledge about car manufacturing, the birthplace of the modern mass-produced car; Detroit in USA (Poppendieck & Poppendieck, 2007, p. 2-5). Henry Ford was notorious for being very proud and very open about his Ford Motor Company so Kiichiro quickly learned how to produce cars "the American way" (Poppendieck & Poppendieck, 2003, p. 1-2). Returning to Japan early in the 1930s Kiichiro spent some years building automobile factories, and in 1936 the first Toyoda car left the factory and success seemed to have no end in sight. But then World War II erupted and effectively put a stop to the dreams of Kiichiro and his cars.

After WWII Kiichiro's company was under pressure due to the gloomy financial state of war torn Japan. The population was poor and demand was scarce. He realized that producing cars "the American way" with mass production was no longer feasible (Poppendieck & Poppendieck, 2007, p. 4). Mass production required a steady stream of raw materials and a stable market with purchasing power, the two things that Japan had none of.

[40] The D in Toyoda was replaced by a T in the car manufacturing company we know today as Toyota Motor Company. Reasons for this vary on the source of the information, some states it as a easier way to write the name in Japanese as it requires two less strokes (Poppendieck & Poppendieck, 2007, p. 3), others that the since Toyota requires exactly eight strokes to writes is it considered as lucky since eight is a lucky number in Japan (Wikipedia.org, 2007)

To cope with this Kiichiro spent the following years working on his vision of "Just-in-Time" production, where no element of the finished product was produced until right before it was needed. In 1962 the Toyota Production System (TPS) was introduced companywide. It more or less singlehandedly catapulted Toyota to become Japan's largest car manufacturer and the second largest in the world (only surpassed by General Motors). Even though the success of Toyota was apparent for all to see their production system was largely ignored by all others in the industrial production world. It was not until the first oil crisis in 1973 that other companies started to look at Toyota. Before the crisis most companies were growing quickly and had little trouble selling all the products they manufactured. The oil crisis changed all that. Almost all larger companies took a hit during the crisis and had to look long and hard at every production they had. One of the only companies that emerged more or less unscarred was Toyota, so naturally all started to look at why that was.

10.3.2 Design more

While the automobile manufactures around the world was flocking to Japan to take a closer look at Toyota and its production system the company also geared a lot of academic interest, mainly from James P. Womack, Daniel Roos and Daniel T. Jones. They had read Taiichi Ohno's book *Toyota Production System: Beyond Large-Scale Production* (1988) and were intrigued by the success of Toyota and wanted to spread the knowledge of the "Toyota Way" to others. In their book *The Machine That Changed the World* from 1990, they laid the foundation of what was to become know in the west as Lean Production and

eventually *set-based design*[41]. The reason why Lean Production became so popular was the rising complexity in the development and design process. More and more departments with different and often conflicting agendas were trying to put their mark on the products, in the hope that their contribution was making the product better.

This often lead to overdesign resulting in complex products that the end-user had a hard time figuring out how to use, or even worse did not fill the need it was indented for. The *affordance* of the design, as Donald A. Norman calls it (2002, p. 9-11), must be apparent for the user to fully understand. Norman's description has gained wide recognition, if fact the term *Norman Doors* has been derived from his book.

Figure 15: How do you open this door? Pull or push the handle?

[41] The term "Set-Based Design" (SBD) was never used in the description of the Toyota Production System, but referred to as a "dynamic model" of the product development process. It was Ward et al. (1995) and later Sobek et al. (1999) that coined the term "Set-Based Concurrent Engineering" (SBCE). Ballard transferred the concept of the set-based approach to the design phase and appropriately named it "Set-Based Design" (2000).

So to be able to cope with all these demands and desires, the design process has to take upon a different approach. This is what the Lean method was designed for. Traditionally development would take on some form of *point-based* design approach, whereas ideas which have to be implemented later will have to confine to the previous developed design. Instead of redesigning all elements to make the new elements fit in their optimal form, the new elements are adjusted to fit the design they are being implemented in.

Doing the design *set-based* is exactly opposite. You delay commitment to one single design and continue work on many different designs. The best combination of the gameplay elements and/ or game mechanics is chosen on the basis on what current moment seems optimal. All additional options are maintained throughout production until final decisions are made. The Lean methods herein the set-based design approach, often strike people as counterintuitive as most other development methods explicitly state that in order to move fast in the development phase you have to make some decisions, lock them and then move on designing the next element. But as Ballard states it is the very notion of developing multiple solutions that gives the team more time for analysis and therefore contributes to an overall better design (Ballard, 2000). As Figure 16 and Figure 17 exemplifies, the difference between point-based and set-based design is not so much in what you make, but in the way you think about it and how you embark upon doing it.

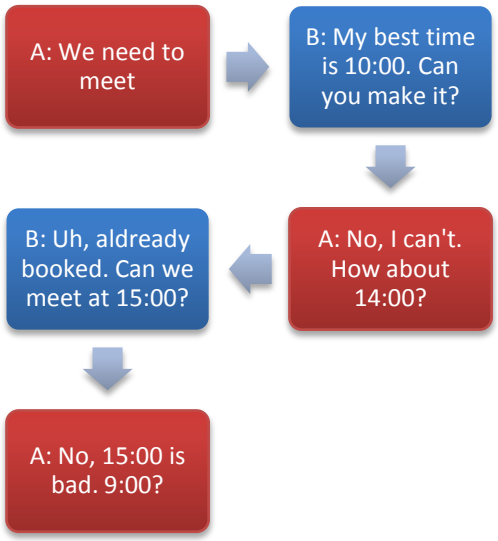

Figure 16: Trying to schedule a meeting with the point-based approach (Poppendieck & Poppendieck, 2003, p. 38).

Figure 17: Scheduling a meeting with the set-based approach (Poppendieck & Poppendieck, 2003, p. 39).

With the set-based approach Person A states from the beginning what alternative he has and Person B then look at her alternatives and finds one that fit with the overall schedule for them both. Person A has to do more work initially but the next step(s) are much shorter and more hassle-free, meaning the overall time spent is shorter. It is all about keeping the design options open. If the designs of specific elements are locked down before complementing elements are designed the later elements have to be retrofitted into the overall design. In connection with game development that is almost always a crippling solution.

Toyota considers a broader range of possible designs and delays certain decisions longer than other auto companies do, yet has what may be the fastest and most efficient vehicle development cycles (Sobek, Ward, & Liker, 1999, p. 68)

Toyota deployed set-based design as the core element of the design process. Instead of deciding on one specific chassis and then fitting the engine and interior elements into (and thereby running the risk of cutting features of the

engine since it might not fit into the chassis) the different engineering departments designed many alternatives and then they in cooperation, decided on what designs complemented each other best. Toyota invented set-based design out of necessity. They wanted to stay competitive and to continuously provide the customers with cars that fit their demands and at the same time made Toyota's manufacturing system flexible enough to cope with shifting markets.

10.3.3 Design more to save money

This meant that Toyota would develop and design a large array of alternative options for each car, which upfront was more expensive and more time consuming but in the long run was extremely time effective and flexible since all elements could be replaced within days/ weeks instead of going all the way back to redesigning the dominating element. The *cost of change* curve takes on a rather different shape as seen in Figure 18. You become less vulnerable towards feedback, both from tests and the surroundings.

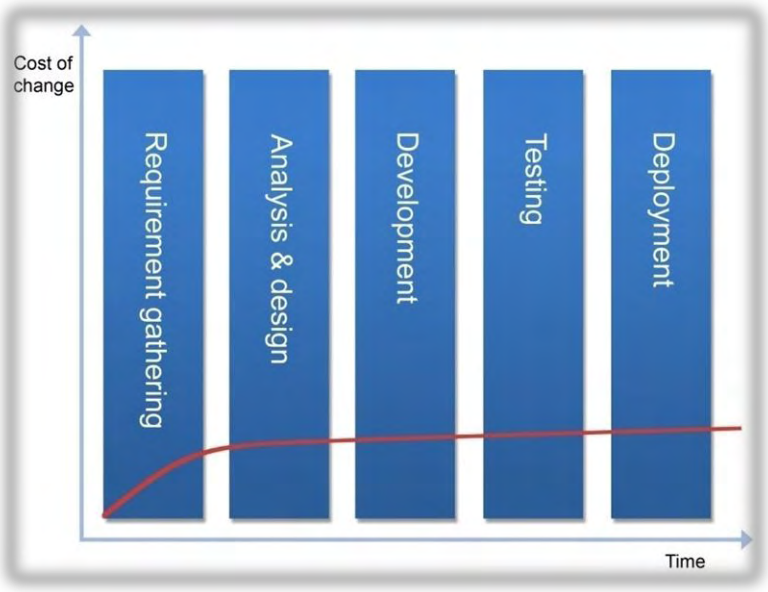

Figure 18: The cost-of-change curve when using set-based design (Beck, 2002, p. 23).

Using set-based design instead of the arguably more manageable point-based design introduces a high cost and high level of workload initially, but leaves the production extremely adaptable for changes later in development. Therefore, it lowers the cost of change significantly in the long term. It does not flatten the curve but makes it rise less dramatically.

Set-based design is exactly about the difference between designing and making. If you are unsure on what exactly to make or how to integrate it all together, the point-based approach will only lead to much rework. New elements that prove not be to working in conjunction with existing design will lead to larger iterations back to previous proven design. It is very hard to make a five course dinner without a recipe for a customer who is not even sure of what he prefers to eat or might not ever have eaten anything ever before. In the *making phase* of development experimentation is not as preferred as it is in the *design phase*. Set-based design is very useful for making "key architectural decisions, that once made, will be very expensive to reverse"

(Poppendieck & Poppendieck, 2007, p. 160), such as fundamental gameplay mechanics in computer games.

10.3.4 Solution Space

Iterations should instead be used to make different possible solutions, and as the design phase progresses the *solution space* should become gradually narrower (Sobek, Ward, & Liker, 1999, p. 79). It is not about *gathering* requirements for your product; it is about *digging* for them. The requirements will often be discovered in during the initial design phases and not in the first project-establishing meetings. The real requirements are very hard to find and are often clouded in assumptions, misconceptions, and politics (Hunt & Thomas, 2000, p. 202). All elements are designed in their respective *design spaces* where they are "free", even encouraged, to explore all possible alternatives to their domain (Ballard, 2000).

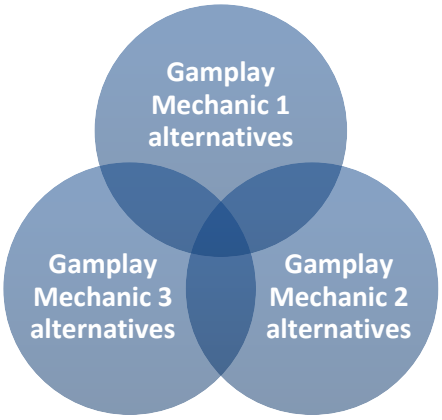

Figure 19: The individual elements and their overlapping *solution space*.

It would seem that designing more options and continuously maintaining them through the design phase is a waste of time and money, but as men-

tioned it makes the designing all the more flexible and open for adjustments. All the options developed and maintained never become wasted work even if they are not used in the final product. They all lead to a larger knowledge understanding whereas all departments gain a greater understanding of the evolving product (Ward, Liker, Cristiano, & Sobek, 1995, p. 49-58). Furthermore, the options developed are ripe for reuse on future productions.

10.3.5 Designing in modules

Set-based design ties closely to what modern software development methods refer to as *modular design*. In most areas of software development, the software product evolves over time, both during development and after initial release. Therefore, it becomes very important how you design your systems in order to prevent unnecessary dual-work.

By encapsulating the gameplay mechanics in independent modules they can easily be replaced, and/ or remove later in the development if some proves to be not working as intended. Using dummy art assets in the lightweight prototypes is an excellent way of speeding up the process of creating these prototypes. If or when better elements are need they can easily be replaced, just

Case study: Modular design

Being able to replace elements of the game proved to be a helpful approach for Monolith the developers of *No One Lives Forever 2* (2002):

"The single most important tool we added was the referential prefab system [...]. In other words, edits made to one file propagate throughout the entire game. For example, if the sound department wants to add a sound to a door opening and closing, they only have to modify a single prefab instead of tracking down every single instance of that game.

The primary advantage of this system is that it puts the power in the hands of the people who need it, without any programmer intervention. A level designer can create a block of geometry that represents a desk, with which he or she can plan the layout of a given room. The art team can then build a nicer-looking desk of roughly the same dimensions to replace the block" (Hubbard, 2003).

like pieces in a jigsaw puzzle. This modular approach has to be planned for from the start of the prototyping phase else it only leads to extensive rework.

Just as Kiichiro Toyoda created the Toyota Production System and laid the foundation for set-based design in order to cope with the harsh market conditions on the Japanese domestic market, so must game developers look beyond traditional point-based design approaches and embrace set-based game design to survive and continuously evolve as the market becomes more and more uncertain and computer games become more and more complex.

11 Testing

> *Playtesting is the single most important activity a designer engages in, and ironically, it's often the one designers understand the least about* - ***Fullerton, et al.*** *2004, p. 196*

Verification and validation with regard to testing is the topic for the present chapter. Earlier we talked about feedback loops and how designers should aim to shorten these as much as possible. We have suggested using the concept of short feedback loops in addition with lightweight prototypes which should allow designers to quickly extract the information needed to decide the future of an idea. Verification and validation should give the answer of how to extract this information. We will introduce the concept of play testing and how this can be used as a tool to acquire feedback from the prototypes.

11.1 Verification and validation

Verification and validation are terms adopted from software engineering which describes the process of checking whether or not a piece of software matches the specification and the expectations of the customer. When developing traditional software this is a process which starts from the very beginning and continues until the product is shipped. The two terms can seem similar, however there is a clear distinction - verification can be explained as the concept of building the product right, while validation is the concept of building the right product (Gold 2004, p. 420).

Verification is the process of checking if the software conforms to the specification – to test if a program runs correctly and delivers the right output. Validation on the other hand is a more subjective approach to check if the pro-

gram meets the expectations of a customer. The two terms serve very different purposes of a development cycle and by so we have chosen to downscale verification in comparison to validation - when developing lightweight prototypes we are not interested in how they are done, meaning the architecture behind, but merely the design and visualization of an idea - is not expected that the prototype itself should be part of a final game. Validation is consequentially more relevant in our context since lightweight prototypes are a simple confirmation test of an idea. A validation test is made to see if the prototype is acting the way we expect, hence making us able to draw a conclusion based upon it. In the next section we will introduce some methods which can be used to perform validation tests in relation to lightweight prototypes.

11.2 Testing in theory

It quickly becomes clear when examining the field of testing that there are many ways to test a product. Usability testing, focus group testing and bug testing are some of the more common methods used when evaluating computer games (Fullerton, et al. 2004, p. 196-197). However, useful these tests might be they serve another purpose than the process of generating and exploring ideas. Each test has a specific purpose – a range of validity one could say and to try to force more from it will only create a disordered result. We wish to introduce the concept of *play testing* (Fullerton, et al. 2004, p. 196). This is a term which is developed specifically in relation to game design, but still shares similarities with test methods used in traditional software development, e.g. *think-aloud tests*[42]. Fullerton, Swain and Hoffman describe the idea behind play testing as the process of gaining "[…] useful feedback from the players in order to improve your game" (2004, p. 196). A keyword in this

[42] A form of testing where the user is asked to say aloud what they are experiencing while using the program.

Case study: Early play testing is crucial

To wait with play testing until the game is done is not feasible and creates problems which are not easily fixed this late in the process. They realized this too late when developing *No One Lives Forever 2* (2002):

"Finally, while play-testing helped balance and tune the game, it should have happened sooner. Thanks to observing play-testers, we made some crucial refinements to the stealth system and the opening missions, but we didn't have sufficient time to play-test the entire game. Play-testing also revealed some design flaws that couldn't be addressed without jeopardizing our ship date. While none of these issues was especially grave, they underscored the need for start play-testing as early as possible" (Hubbard, 2003).

definition is *players* - computer games separate themselves from traditional software in the sense of not knowing who the end-user is. When developing a banking system the end-user is known and it is valid to assume that all bank employees will act in the same way when using the product. This however is not the case with computer games. It is impossible to generalize the end-user when talking about computer games.

So who is the player when we want to test our computer games? Most of the agile development methods refer to what they call the *customer* or *product owners*[43] as end-users. These methods suggest the use of end-user participation in the development of the product. This is from the viewpoint of being able to quicker and more precisely adapt to what the users (really) want and demand instead of doing more formal and rigid testing towards the end of the production cycle. Having end-users sitting together with the development team in game development is much harder and do not make as much sense as it does in software development. The fact that most computer games have a rather generic target-group in comparison to for example a booking system for a travel agency makes it harder to receive objective feedback since it is based on whether or not the game is fun. Furthermore, the information re-

[43] The customer and product owner terms are from *Extreme Programming* (Beck & Andres, 2005, pp. 61-62) and *Scrum* (Schwaber & Beedle, 2002, s. 34) respectively, but in broader terms they cover the same ground.

ceived from the use of actual players is everything that was popular 10 minutes ago and how this stood out (Cerny & John, 2002). Players have a tendency to focus on computer games already known to them consequently the important things are elements which already exist. Since we aim to create new ideas through explorative design this is not a viable way to test the lightweight prototypes.

Therefore, instead of having an onsite end-user sitting side by side with the game development team, the team often must rely on prior experience, knowledge, and old-fashioned gut feeling. The game designers then act as customer proxies (Larman, 2004, p. 152-153) trying to put themselves in the place of end-users. They ask "stupid questions" about the design and in general try to scrutinize the overall game development. Software development specialist Steve McConnell[44] writes in his book *Rapid Development;*

> Putting yourself on the same side as the customer is one of the best ways to avoid the massive rework caused by the customer deciding that the product you spent 12 month on is not the right product after all (McConnell, 1996, p. 16)

The idea of using the designer as an end-user also fits the concept of lightweight prototyping. The use of prototypes is a rapid and dynamic way of working. Every idea is explored in timeframes of a few days, even hours if possible which makes it unfeasible to engage in big formal and structured tests with players. It is not a process were a single prototype needs testing but a process were all important game mechanic needs to be tested via prototypes. If a design team were to make formal test scenarios for all of these it would be a time and resource consuming task to do. Fullerton, Swain and

[44] According to website Wikipedia.org Steve McConnell was named one of the three most influential persons in the software industry in 1998 along with Linus Torvalds and Bill Gates. See http:/ / en.wikipedia.org/ wiki/ Steve_McConnell.

Hoffman explain the process of using the game designer to test with as *self-testing* (2004, p. 198). The idea is that a lightweight prototype is constantly evaluated during development by repeatedly playing it. Self-testing can be compared to what Schultz, Bryant and Langdell refers to as *ad hoc* testing where the designer daily goes through a series of thoughts like "I wonder what happens if I do…?" (Schultz, et al., 2005, p. 287). These simple thoughts form the basis for the self-testing and through this knowledge is gained regarding how the game mechanics work and what can be done to improve them. It is a process which fits the early phases of a development best but can continue throughout the whole process. The danger of this is that the vision imagined is not as clear to others as it is to oneself which can lead to a one sided design. At some point it can be worthwhile showing it to others in order to identify flaws or improve the design. These can be colleagues, friends or other confidants who can view the prototype with fresh eyes (Fullerton, et al., 2004, p. 198). The benefits when using such people is that they are probably familiar with the project you are working on and by so a thorough introduction to the game is not needed. Nevertheless what this approach has in simplicity it lacks in objectivity. Using friends, family or colleagues require a lot of discipline in regards to feedback. Too much optimism or pessimism is often the result when letting such people review the work done. This can distort the conclusion and give an inaccurate picture of how the prototype in fact works. To be mindful of this is important and questioning the feedback given can uncover what the participants really think about the prototype. The more that is known about the prototype the better chance there is to make a qualified judgment of its future.

11.3 Practical approach

In the following section we will try to give a more tangible approach to how testing of lightweight prototypes can be done. Even though the play testing explained in the previous section might seem fairly simple there are some issues to be aware of when conducting self-testing and tests with confidants.

Self-testing can be a challenge where the designers easily get blinded by their own vision. This is at least what we experienced during our experiments. A "pat on the back" mentality can easily evolve and instead of looking with critical eyes on what you are doing you end up gratifying yourself. On the other hand over-criticism can also destroy the best ideas. A solution to help find a balance between optimism and pessimism can be to ask some simple questions – "what is the objective of the prototype?", "can I think of another solution?", "is object X doing what I intended it to do?" etc. Such questions can both act as help to develop the prototype and give a more tangible element to measure the prototype against – did it perform as expected?

The use of confidants is a way of working which can be used in many situations. It can be an informal talk at the vending machine or it can be a more structured meeting where the prototype is presented. The prototypes are very much a work in progress and to constantly ask colleagues to comment on the

Case study: Play testing reveals design errors

Play testing is the only way to find flaws and errors in the design. The people behind *Fireteam* (1998) used play testing as a tool to improve certain aspects of the game:

"I can recall a particular controversy over whether Gunball (a Fireteam scenario similar to combat football) was balanced enough. Many of the advanced players were complaining that Gunball's offense was too hard. Using our Tile Edit tool, we quickly created a few maps with two endzones for each team (Gunball maps normally only have one endzone). Through testing the new maps, we discovered some of the problems with Gunball were unrelated to the maps themselves, but that the offense simply had a disadvantage when trying to score. So instead of redoing all of our map designs, we tuned the Gunball game by giving the Gunball carrier a protective drone" (Min, 2000).

work done is both irritating for them and the designer. What we did during the experiments was to schedule a weekly meeting where people got a chance to play and comment on the prototypes made that week. The person(s) behind the prototype would give a brief introduction whereafter each participant would get the chance to play. This worked very well and created a lot of feedback which each person could use to improve or change the prototype with. When conducting such a meeting a series of questions can help as a starting point for the discussion. Questions such as "*is the game mechanic too easy?*", "*are the controls intuitive?*", "*is the game mechanic easy to learn?*" are all examples that can help answering the ultimate question in regards to play testing – "*are the game mechanics fun?*" (Schultz 2005, p. 296).

A way of thinking which we adopted from Extreme programming was the use of pair programming[45]. Not in the sense that we would sit two together and code but more the idea of having a team member which could evaluate the work done. We experimented with different team constellations and found that even though one person would create the most special but at the same time most single minded prototype, teams of two or four would create an overall better prototype with more uniform ideas. The continuous discussion and evaluation in the team was the key element to improving the ideas. The problem then became that the explorative element in lightweight prototyping was lost – it is a fine balance between exploration and evaluation. Four person groups are definitely too big and resulted in too much polish. The crazy ideas were often removed from the prototypes since everyone had to be pleased – it was more a compromise than an explorative prototype. Groups of two worked well in the sense of exploration and at the same time gave the possibility for evaluating the work done.

[45] The idea is to sit two persons at one computer and write the program code together. This should lead to better and more reliable code, since they can work together and continuously evaluate the program. (Beck 2005, p. 42)

A crucial part of developing lightweight prototypes is to use a development tool which allows the idea to be designed and evaluated quickly. During our experiments we used a tool called Game Maker[46] which allowed us - with a single click on a button – to quickly make an executable version of the prototype. By doing so we were able to make frequent tests and evaluate the design. The important thing is not whether or not to use Game Maker but to use a development tool that you are familiar with and allows you to make frequent builds – waiting for a program to compile for hours is not feasible with regard to lightweight prototype testing.

11.4 Summary

The important lesson to remember about play testing is that it can never been done enough. To neglect testing can have serious consequences for the product if basic elements of the game do not work probably when released. Furthermore, to use big formal test sessions in relation to lightweight prototypes is not a viable solution. Instead the methods used have to be an integrated part of the design process. The process of self-testing and testing with confidants is easy and cheap ways to test if the essential gameplay works as intended. Both methods support the idea of short feedback loops and flexible design as a way to work explorative. Play testing is the primary source to receive the information needed when deciding how to improve or change a lightweight prototyping.

[46] Game Maker is a basic drag-and-drop program for creating games. It allows you to make everything from simple platform games to advanced 3D games.

12 The EVE Method

So far we have been looking at existing development methods in several different fields. We have discussed software engineering, innovation and game design theory in relation to game development. We also had a look at post mortem articles from game productions, and went on to identify some of the problems game productions are currently facing. In doing so, we found that the single largest source of problems is the pre-production phase and the process of verifying new ideas. This is where our method comes in. We want to give developers a set of tools to help them refine their ideas for game mechanics, a framework which at the end of the day makes the end product better and production run smoother. A small change in the beginning of the project can be all it takes, and in this chapter we will look at one way of making this change. We have called our method *EVE*: *Experimentation*, *Visualization* and *Evaluation*.

One of the main goals of our method is *knowledge gathering*. The more you know about an idea the better your decisions will be, and we believe in a hands on approach. In part, this knowledge gathering is done through early experimentation. The idea behind early experimentation is to look at an idea from different angles and through a refinement process of trial and error give the designers a better understanding of their own ideas. This can help reveal flaws in the design at an early stage of the process, saving considerable work compared to discovering them later on. Assuming that the initial idea is perfect is a high risk business, and experimentation helps reducing this risk. For this reason, creating a safe setting in which mistakes can be turned into cheap and valuable lessons is the first step towards implementing the EVE method.

Once that mindset is in place, it is time to start visualizing your ideas. While every idea usually starts on a piece of paper the written word will only take

you so far. Nobody would even consider starting a game production without first making some concept art and the same should be true for prototyping game mechanics. That is why we suggest making *lightweight prototypes* as a way of giving designers an idea of how their ideas will feel when implemented in the game. A playable prototype makes it much easier to convey the idea to the rest of the development team, and the process of creating prototypes also gives the designers a better understanding of their own ideas. In addition, a playable prototype makes evaluation possible.

By *evaluation* we mean that constantly evaluating your own work as well as getting evaluation from others. By creating prototypes and experimenting with different solution, the designers create something tangible for themselves and their team. Simply playing around with your own prototype can expose weaknesses in the design, and having other developers from the team try them can further help with this process. Another goal of EVE is to create a feedback loop for new ideas, where feedback is given as soon as possible so it can be taken into consideration. We have seen from post mortems that many developers went 6 months or maybe even a year into production before they had something playable up and running. We want to turn this trend completely around.

This method is designed for testing out new ideas, whether that happens in the pre-production or during the production. The thing to remember is that it is possible to evaluate elements of the game before the game is completely finished, and even before the design is finalized. The kind of prototypes we suggest can be everything from different colored dots moving around on the screen, testing A.I. behavior, to a fully functional inventory system implemented in the engine. As long as the process from idea to playable prototype is fast, the tools used to build the prototype does not matter. In pre-production the engine which will be used for the finale game might not be

available or sufficiently understood, and in those cases it is better to use an existing engine to illustrate the different mechanics. Later in the process when a skeleton of the game is up and running, making something in the engine might be just as easy as using a different tool. As long as the prototype can be made in a few days, the tools used for making it are irrelevant. But there is one important point to remember; prototype code should not be reused in the final code.

If you are taking the time to make your prototype code of publishable quality, chances are you are wasting your time. This is the arena for hack solutions, shortcuts and dirty fixes. The goal here is quickly make something you can use to illustrate your game mechanics, not something you can copy-paste into the latest build. There is an additional pitfall here; the *more* time you spend polishing your prototype, the *less* likely you are to trash it if it turns out not to work. This concept scales perfectly: throwing away two days of work is easier then throwing away two weeks' worth of work. Nobody wants to throw away half a year worth of work, and that is what our method is trying to save developers from doing.

However, some game genres will benefit more from this approach than others. For instance, strong story based genres like the adventure game will benefit less from this approach, yet it could be used to test mini-games and the usability of the interface. EVE is a method for testing *new* concepts and ideas for game mechanics, if a game's main selling points are graphics and story there is less to be gained from using this approach.

In the following section we will try to elaborate and clarify what the three stages of the EVE method involves. EVE is more a mindset than a list of tools that are directly applicable in a game development. Lastly we will exemplify the method through a story which should give an idea of how to use the method in a more practical approach.

12.1 Experimentation

Failure is an option - the possibility of failure is something which every project has to deal with. Schedules, models, specifications etc. are all tools to minimize the risk of failure. However, precise as these may be it is still impossible to predict every situation that can occur in a production. The best way to prevent these situations in a production is to have tried them before doing it for real. So how do we do that? We suggest a way of working were you test the essential and cost expensive parts before you go into full production. By doing it in a much smaller scale it becomes less risky and more manageable to make errors. To cancel a project in full production can lead to economical disaster, so to testing the vital parts of a production in an environment which is both cheap and allows failure might end up saving the company a lot of money.

One thing is to be aware of failure; another thing is to embrace it. The essence of a preproduction phase is to test all the crazy ideas in an environment where failing miserably are indeed an option. To embrace failure in preproduction is to be financially responsible for the much more expensive production that follows. If you can weed out all the bad ideas by testing them in a cheaper preproduction you will save the effort and money when 50 people depends on your work in a later stage of the production.

Exploration as a design tool – it takes a thousand bad ideas to find a good one and to explore the bad ideas will help you understand the good ones. Exploration is easier said than done – it requires a lot of discipline and creativity to explore alternative solution to a problem you already solved. To explore different solutions is like searching for a proof of concept. You might find the perfect solution or you might not, but no matter the outcome the time spent should give you more information and by doing so increase your chances of making the right decision.

Set-based design is a way of thinking where the explorative element is an on-going process. You keep developing multiple prototypes of the same ideas were the one which fits the overall design best will be the one used. This way of working ensures that you always have a game composed of elements which work together as one unit. If something ceases to work in the context of the game it can simply be replaced by another element which have been tested and evaluated by means of prototyping. To try many possibilities without any prejudices is the key element when working exploratively.

12.2 Visualization

Playing is believing - one picture says more than a thousand words, which is the concept of visualization. Computer games are not something you can easily imagine or explain. You have to try it and in order to do so you have to build it or parts of it. What might seem fun on paper can turn out to be tedious when playing it. Our intention is to introduce the tangible element in game development as soon as possible through the concept of *lightweight prototypes*. The use of these makes it possible to clarify by means of playing if and how an idea works in relation to the overall vision. They act as a small confirmation test from which you can draw information on whether or not the idea is worth pursuing.

Lightweight prototypes does not mean first playable – the term *prototype* is used in many situations throughout the game industry and have different meanings. However where these often represent a bigger fraction of the finale game, a lightweight prototype is made to illustrate the smallest parts of a game – *game mechanics*. Prototypes such as first playable, vertical slice, alpha, etc. are all an important part of a game development, however they serve a different purpose than a lightweight prototype and when compared the

amount of time and resources spent separates them clearly from lightweight prototypes.

The production timeframe of a lightweight prototype should be no more than a few days, even hours if possible, since the idea is simply to gain enough knowledge about the idea to be able to make a sound judgment regarding the future of it.

Keep it simple – lightweight prototypes are not just a way of working but also a way of thinking. It is easy to make things more complex, but the point of making a lightweight prototype is to illustrate the basic idea. The urge for doing beautiful graphics, animations, story etc. must be downscaled, since you only risk wasting valuable time at this point. That does not mean you should exclude everything that makes a prototype look great – of course not. However, looks or other features must never be prioritized over the game mechanics. If an idea cannot stand on its own without cool graphics or impressive sounds, it probably is not such a good idea anyway.

Single out the key elements - What is the core of your game? What do you want the player to experience? Why is this element important for the game? These are all questions which can help you find the core of your game – the game mechanics. Before the prototyping phase can start a clear understanding of what these are must be in place. The use of keywords, brainstorming and "the six thinking hats" method are all ways of working which can help you find and understand the game mechanics. If you end up with multiple game mechanics – which will probably happen - prioritize them after level of importance and use this as a starting point for the visualization phase.

When doing lightweight prototypes, keep the scope at a manageable size. Each game mechanic should get its own set of prototypes, and trying to prototype several mechanics at once will risk clouding the output. It is easier to

test the idea if the variables are kept to a minimum. Make many small prototypes instead and test them individually. This should give a much clearer view of which ideas that could work in a final game and which ones should be scrapped, and once tested the good prototypes can always be combined later for further testing.

Abstract prototypes - a reason for using prototyping is to quickly reach a point where something playable provides a clear picture of the vision. Using a lot of time on fine tuning certain elements and details like graphics and sounds will only increase the impression slightly - a red dot which takes 1 minute to make can at this point work fine in contrary to using an animated character which took you numerous hours to make. One could argue that it is the details which differentiate great computer games from bad ones. True, however, it is testing and evaluation which is the goal of the visualization process, so save the details for the final game and instead use the time to explore other facets of the idea. Roughly said the process of making prototypes is about quantity and not quality – to explore the different possibilities by means of many prototypes instead of using time polishing a single one.

Modular prototypes – the success of a game production depends on how well you handle changes. Even small developments will face the need to change something at some point so you can either plan for it in advance or face the problems as they emerge. The use of prototyping is a perfect way to plan for changes. The idea is to think of lightweight prototypes as Legos; each prototype is a module that when, combined with others, is united to form something greater. Each game mechanic is tested through numerous prototypes where each implements a different version of the same mechanic. When combining the ideas in a final game it becomes easy to exchange one mechanic for another if you realize that it does not fit or the design has been change

in some way. Modular prototyping is about having more red Legos ready if the one you tried first does not fit your house.

12.3 Evaluation

Play testing is a never-ending process – the first step to improve something is to understand where and why the flaws are and the only way to do this is through evaluation. The process of using lightweight prototyping is a dynamic and ongoing development of ideas which consequentially limits the possibility for formal and structured testing sessions. Instead testing must function as a natural part of the design process. Self-testing is a way to approach evaluation where playing the prototypes should help to understand what and why something is done wrong. Through this continuous testing you gain a broader understanding of the idea and the context in which it functions. To ask yourself questions concerning the idea and the relation to the rest of the game and to compare it to similar prototypes will help you see new perspectives or even whole new solutions to the problem.

However the most important way to gain an understanding of the idea is to have other people evaluate the prototype. Feedback is by far the most rewarding way to evaluate anything and neglecting it will only hurt the game. The use of confidant-testing fits the lightweight prototyping approach were colleagues or other confidants act as test persons. But it is important not to make this into a formal test session; at this point such an elaborate setup will be a waste of time. Instead, invite a colleague in for a cup of coffee while you show your work and simply make a mental note of the things he or she says and does relating to the prototype. You will go back to working on the prototype the minute your colleague leaves anyway, so there is no need to document the feedback more extensively then on a Post-it note. All information is

welcome at this point, and you can always dismiss irrelevant comments later on. Remember, you are not building a game at this stage, but simply exploring ideas, so to receive all the feedback possible will only increase your chances to build a great game later on. Furthermore, the input from others helps you to see new sides and by doing so prevents your ideas to be single-minded.

Make throw-away prototypes – so finally you managed to make a really great prototype which is ready to be implemented in a final game. However tempting this might be it is important to stress that this should not be an option. Do not expect your work to be used for anything else than evaluating the concept. The prototype should be nothing more than a source to extract information from. When making prototypes there are no rules of how to make them and there should not be since the goal is to get something up and running as soon as possible. To demand that the work done can be used in a final game only limits the creative process and makes people think about issues such as implementation, appearance and reusability which are not important at this stage – when doing lightweight prototypes it is always more important to focus on *what you made* instead of *how you made it*.

This way of thinking is something which needs to be adopted by the entire development team in order for lightweight prototyping to work. Appearance and eye-candy is an effective way to sell products that lacks quality themselves and to promote this as an important feature when prototyping removes the focus from game mechanics and onto something which this process is not intended to be used for. To make lightweight prototypes is to accept the absence of high value concept art, setting, and sound and to focus on game mechanical issues exclusively.

Be proud of your work – in order to receive feedback you need to show your work to others. While this sounds good on paper, it is easier said than done.

To show a prototype in which you have spent valuable time implementing a couple of dots that moves around does not seem as a good way to impress your boss. However, for a developing team using lightweight prototyping it is important to promote this behavior. If not, you risk that people start prioritizing visuals over game mechanics in order to "sell" their product better to colleagues. When working with lightweight prototypes it requires a show-off element among the developers for the process to work – everyone has the same goal and everyone should appreciate the value of a lightweight prototype.

Make decisions as late as possible – the information regarding game mechanics may be somewhat limited in the beginning of a production and to base the entire game design on these may turn out catastrophically. Instead of making important decisions at such an early stage of the production, the rational thing would be to wait until you know more. Evaluation of lightweight prototyping should be the key element in obtaining the knowledge needed in order for you to make a reasonable decision. Does it take one, five or ten prototypes to give me the information I seek? Well, the answer is of course not black or white, however, a rule of thumb is to keep prototyping until you simply cannot wait any longer to make a decision. The idea is to make important decisions regarding game mechanics at the last responsible moment since you might gain new information which changes the context and requires a new decision, making previous decisions a waste of time.

12.4 A day in the life of a rapid prototyper

An idle Tuesday morning, John is pouring his first cup of coffee for the day. As a designer on the company's latest project he's got his work cut out for him, and the expectations are high. Walking to his desk he runs the combat

mechanics over in his head. They are still only a rough sketch, but the basics for the third person shooter are there.

Coffee in hand, John heads for his workstation. But before he gets there, Peter, one of the other designers, grabs his arm - his latest prototype is ready and he needs some feedback on it. Peter's prototype is simple; a box moving through a blocky environment and a movable crosshair corresponding to the Xbox controller which has been hooked up. Pressing 1-5 on the keyboard cycles through different possible control mappings, allowing for a comparison later on once more pieces are in place. They discuss the prototype for a few minutes before John returns to his workstation.

On the wall in their office hangs Post-it noted describing different game mechanics, ranging from inventory system features and reload mechanisms to special AI behavior and suggestions for puzzles. John pulls down the note named "active reload", reading the two line description as his computer boots up... "Make reloading more interesting by adding an element of timing". He starts working on a small prototype, spending the first half of the day experimenting with different ways to implement this element of timing. Throughout this process all the designers are there to provide each other with feedback once something is ready. As soon as something playable is up and running, one or both of the other designers spend a few minutes trying it out and commenting on it. Once the prototype is good enough to explain the idea, a short description of the prototype is written in the wiki along with a link to the file. Their wiki allows everyone to get a feeling for how the project is coming along. The publisher can see which direction the game is tanking, and people on the development team can try the different prototypes and comment on them.

13 Conclusion

This project will outline a method based on proven practices which can enhance and support the process of creating, exploring and evaluating ideas in the context of game development.

The computer game industry has during the last decades experienced a steady increase in both developers and the overall production cost of their computer games; we want bigger and more complex computer games to entertain ourselves with. The demand for delivery on time from investors and publishers has now forced the developers to search for structure and management when developing computer games. The possibility of failure is a constant fear within computer game development since the amount of money at risk has simply become too high. The parallel world of software engineering has faced many of the same problems which we see in the computer game industry today. Here, the solution to better products and working conditions came in the form of methods and practices. They managed to tame the wild beast of creativity by means of structure which ensured a viable environment to develop software in. Even though we see some of the same tendencies in computer game development regarding structure and process management there is still a great leap from the way game developers approach their products compared to software developers. Many aspects of a computer game development can be compared directly to aspects of software development – elements like implementation, functionality and usability is all key concepts in both areas. However, the reason why the process of software development cannot be directly adopted by game development is due to the *fun factor*. This inherent quality of a game separates it from traditional software by bringing a subjective element into the development process. The fun factor is depended

on the individual person and is not directly measurable which creates a different setting for a possible method.

We have during this project developed a method which takes the fun factor into account. We have through an analysis of the game development industry identified several problem areas, such as lack of flexibility, feedback and vision as a result of an unstructured pre-production, which led to an inadequate design and consequently made the problems escalate throughout the actual production. We have argued that a game development when going into full production converts from being a creative and explorative process to one comparable to a software development process. This along with the problem areas suggests that the point of insertion for a change in practices is in the design and conceptualization process where the foundation for the entire game should be created and explored and where errors left unattended will damage the ongoing production.

The EVE method is our answer on how to structure the design and conceptualization process. We have tried to address the problems found in the analysis through proven theory and actual practices. Theory from software engineering along with innovation theory has been the source of inspiration on how to solve these problems and our experiments has been the tool to understand the theory. EVE stands for *Experimentation*, *Visualization* and *Evaluation* and offers a variety of practices and tools to help the game designer gather knowledge. This makes him able to draw a conclusion in regards to the idea - is it worth pursuing? The method should not be seen as a sequential process but as a unit where you use and cycle between the three phases as required. Furthermore, the method has a specific range of validity; it was develop with the purpose of exploring and testing *game mechanics*.

The three elements of EVE can be explained as:

Experimentation: To explore all potential solutions and to accept the possibility of failing when doing so. By exploring an idea at an early stage you remove risks which could potentially result in disaster later on in the production. *Fail early and succeeded later* is the mantra to remember when experimenting.

Visualization: To convert the idea from thought to a more tangible medium. By using *lightweight prototypes* you are able to create something concrete which can help to understand the idea better. Furthermore, these will help convey and communicate the idea to the rest of the production team. Computer games are all about playing, so the design should become playable as soon as possible.

Evaluation*:* To play test the prototype in order to reveal flaws or find improvements. Evaluating the prototype through *self-testing* or *testing with confidants* will give you a better understanding of which mechanics that work and which that do not. The intention of the evaluation phase is to create short feedback loops, thus improving both the prototypes and the ongoing production.

We have used the term *method supplement* to explain where and how the EVE method fits into a production. It is not a whole method by itself but more of an add-on to existing methods and should be seen as a way to improve or change the way a design process is done. We have designed it with agile development methods in mind – it embraces and uses the principles from these.

To sum up we have developed a method which builds on principles of experimentation, visualization, and evaluation which takes the fun factor of a game into account by means of creative process control, flexible design in form of prototypes and easily accessible feedback through early testing. This

should let you, as a game designer, focus on the important aspect of a game – the game mechanics.

14 Future research

In this chapter we will take a look at what could have been done to make the project take on another point of view, as well as how our report can be used as a starting point for future research.

Using the post mortem articles we were able to look at game productions from the past 8 years in order to get a little inside information about some of the problems these productions have been facing. This could be taken a step further by doing a deeper, more quantitative analysis of these articles and try to find more subtle differences between the different productions. Alternatively, a more qualitative analysis of the industry could be made through interviews with several different developers from different companies. Both these options could be the starting point for an entire project studying the field of game development, and where our research have been goal oriented a project like that would be more of a more explorative nature. Furthermore, an inquire like that could be expanded to look at cultural differences between American, European and Asian game studios, development over time, or even development within the big successful game studios like Blizzard or EA Games which have been making computer games for over a decade.

A completely different angle would be to look closer to other fields than software development. We have not been looking at TV and movie productions in this project and there might be some things to learn from those kinds of entertainment productions. While lacking the interactivity element from computer games, there is still a consumer which wants to be entertained in one way or another. Experiences with gathering feedback on new movie concepts might very well be useful for game studios working on new productions, beyond the aforementioned focus group.

As far as testing goes, our own testing the method have purely been on a prototype level were we explored the difficulties attached to this way of working with new ideas. The real test of the method is to implement it in at an existing game studio and follow the process closely. This could be done either by including it as a way of thinking during the pre-production, or it could be incorporated into the work method of designers working on projects which are already in production.

The method we have proposed is primarily targeting the concept and pre-production phase. This could be used as a starting point for describing in detail a whole process built on the mentalities we describe. For such a project, our report could serve as a foundation on which to build a more extensive and complete method describing a game production from start to finish.

16 Appendix B – Postmortem source material

Release date	Title	Developer	Publisher	Review rating	Budget	Development time	Developers	Contractors	Code length
March 1998	Sanitarium	DreamForge	DreamForge	84		16	37	0	
June 1998	X-Files	Fox Interactive	Fox Interactive	63		48	30	3	
October 1998	Trespasser	DreamWorks Interactive	DreamWorks Interactive	56	$ 7.000.000	36	39	0	
November 1998	Heretic II	Raven Software	Activision	84	$ 1.100.000	11	32	0	
December 1998	Thief: The Dark Project	Looking Glass Studios	Eidos	90	$ 3.000.000	30	19	5	
December 1998	Fireteam	Multitude	Multitude	75	$ 2.500.000	30	14	3	
June 1999	Descent 3	Outrage Entertainment	Interplay	85	$ 2.000.000	31	19	0	
July 1999	Heavy Gear 2	Activision	Activision	82		19	20	0	
August 1999	System Shock 2	Irrational Games	Looking Glass Studios	92	$ 1.700.000	18	23	0	
August 1999	Draken: Order of the Flame	Surreal Software	Psygnosis	81	$ 2.500.000	28	23	2	
September 1999	Command and Conquer: Tiberian Sun	Westwood Studios	Electronic Arts	80		36	33	0	
October 1999	Age of Empires II: Age of Kings	Ensemble Studios	Microsoft	91		24	40	0	
November 1999	Star Trek: Hidden Evil	Presto Studios	Activision	53		12	22	0	
November 1999	SWAT 3: Close Quarter Battle	Sierra Studios	Vivendi Universal Games	84	$ 2.200.000	18	20	0	
November 1999	Unreal Tournament	Epic Games	GT Interactive	93	$ 2.000.000	18	16	0	350000
November 1999	Resident Evil 2	Capcom	Capcom	87	$ 1.000.000	12	9	1	200000
November 1999	Gabriel Knight 3	Sierra Studios	Sierra	79	$ 4.200.000	36	45	3	350000
March 2000	Soldier of Fortune	Raven Software	Activision	82		23	20	2	406044
June 2000	Vampire: The Masquerade -- Redemption	Nihilistic Software	Activision	76	$ 1.800.000	24	12	8	366000
June 2000	Diablo II	Blizzard Entertainment	Blizzard Entertainment	88		30	40	0	
June 2000	Deux Ex	Ion Storm	Eidos	91		34	20	4	
July 2000	Heavy Metal: F.A.K.K. 2	Ritual Entertainment	Gathering of Developers	79	$ 2.000.000	18	18	1	364825
September 2000	Star Trek: Voyager—Elite Force	Raven Software	Activision	86		24	20	13	919749
November 2000	Hitman: Codename 47	Io Interactive	Eidos	73	$ 3.000.000	36	36	0	
November 2000	No One Lives Forever	Monolith Productions	Fox Interactive	89		24	18	0	110000
March 2001	Fallout Tactics	Micro Forte	Interplay	81		18	27	0	300000
March 2001	Black & White	Lionhead Studios	Electronic Arts	90	$ 5.700.000	37	25	3	2000000
April 2001	Tropico	PopTop	Gathering of Developers	82	$ 1.500.000	24	10	1	170000
June 2001	Startopia	Mucky Foot Productions	Eidos	85	$ 3.000.000	24	19	2	335000
June 2001	Operation Flashpoint	Bohemia Interactive Studios	Codemasters	86	$ 600.000	50	10	3	250000
September 2001	Trade Empires	Frog City Software	Eidos	69		15	9	3	
October 2001	Dark Age of Camelot	Mythic Entertainment	Mythic Entertainment	87	$ 2.500.000	18	25	5	
October 2001	The Italian Job	Pixelogic	SCI	58		15	9	0	
November 2001	Cel Damage	Pseudo Interactive	Electronic Arts	67	$ 2.000.000	24	16	12	800000
November 2001	Star Wars Rogue Leader: Rogue Squadron II	Factor 5	LucasArts	90	$ 3.500.000	9	30	2	
December 2001	Jak & Daxter: the Precursor Legacy	Naughty Dog	Sony	90		36	35	0	
January 2002	Medal of Honor: Allied Assult	2015	Electronic Arts	91		19	27	2	469644
January 2002	Draken: The Ancients' Gates	Surreal Software	Sony	80		30	30	5	326000
April 2002	Freedom Force	Irrational Games	Electronic Arts	89		18	25	1	
April 2002	Spider-Man	Treyarch	Activision	78		18	40	0	300000
April 2002	Dungeon Siege	Gas Powered Games	Microsoft	85		44	27	5	800000
May 2002	Aggressive Inline	Z-Axis	Acclaim	85		14	25	2	
June 2002	Neverwinter Nights	BioWare	Atari	89		60	75	65	
October 2002	No One Lives Forever 2	Monolith Productions	Vivendi Universal Games	91		19	21	0	150000
October 2002	Age of Mythology	Ensemble Studios	Microsoft	90		30	50	10	1500000
October 2002	Gothic II	Piranha Bytes	JoWood	79		11	13	40	
October 2002	Hitman 2: Silent Assassin	Io Interactive	Eidos	86	$ 7.400.000	23	70	0	
November 2002	Ratchet & Clank	Insomniac Games	Sony	89		18	40	1	
December 2002	Big Mutha Truckers	Eutechnyx	THQ	66		24	40	0	
January 2003	Battle Engine Aquila	Lost Toys	Atari	74		30	16	2	380000
March 2003	Splinter Cell	Ubisoft	Ubisoft	88		5	76	18	
March 2003	Amplitude	Harmonix	Sony	85		15	20	7	380000
March 2003	Jurassic Park: Operation Genesis	Blue Tongue Interactive	Universal Interactive	70		22	26	0	
May 2003	Rise of Nations	Big Huge Games	Microsoft	89		30	26	16	837939
July 2003	Star Wars: Knights of the Old Republic	BioWare	LucasArts	94		30	70	28	
August 2003	T.R.O.N. 2.0	Monolith Productions	Buena Vista Interactive	84		24	24	0	853300
September 2003	Homeworld 2	Relic Entertainment	Vivendi Universal Games	85		39	30	5	
October 2003	Jak II	Naughty Dog	Sony	88		24	48	0	841000
October 2003	Freedom Fighters	Io Interactive	Electronic Arts	82	$ 6.700.000	30	110	0	
November 2003	Secret Weapons over Normandy	Totally Games	LucasArts	80		18	26	0	246513
November 2003	Project Gotham Racing 2	Bizarre Creations	Microsoft	93		24	40	62	219538
November 2003	Prince of Persia: The Sand of Time	Ubisoft	Ubisoft	92		27	65	0	1263580
March 2004	The Suffering	Surreal Software	Midway	81		26	37	23	
April 2004	Hitman: Contracts	Io Interactive	Eidos	77	$ 12.000.000	12	110	0	
June 2004	Shadow Ops: Red Mercury	Zombie Studios	Atari	59		24	37	0	
July 2004	Spider-Man 2	Treyarch	Activision	81		24	60	0	
September 2004	Kohan II: Kings of War	Timegate Studios	Take Two	81		27	35	0	
September 2004	Katamari Damacy	Namco	Namco	86		18	21	0	
September 2004	The Sims 2	Maxis	Electronic Arts	90		42	140	0	
September 2004	Silent Hill 4: The Room	Konami	Konami	76		30	70	0	500000
November 2004	Axis & Allies	Timegate Studios	Atari	65		22	39	0	
November 2004	Ratchet & Clank 3: Up Your Arsenal	Insomniac Games	Sony	91		18	65	2	2000000
November 2004	Alien Hominid	The Behemoth	0-3 Entertainment	78		18	12	0	780000
April 2004	Half-Life 2	Valve Entertainment	Vivendi Universal Games	96		71	40	15	
December 2004	Star Wars: Knights of the Old Republic 2	Obsidian Entertainment	LucasArts	86		15	33	0	
January 2005	Resident Evil 4	Capcom	Capcom	96		30	80	0	
April 2005	Psychonauts	Double Fine	Majesco Entertainment	87	$ 11.800.000	54	42	5	592009
September 2005	Farenheit / Indigo Prophecy	Quantic Dream	Atari	84		24	80	0	
October 2005	Stubbs The Zombie	Wideload Games	Aspyr	75		17	12	56	600000
November 2005	Guitar Hero	Harmonix	Red Octane	91		9	49	0	335100
November 2005	Gun	Neversoft	Activision	77		24	60	0	
November 2005	Peter Jackson's King Kong: Official Game of the Movie	Ubisoft	Ubisoft	80	$ 15.000.000	24	280	0	
April 2006	Tomb Raider: Legend	Crystal Dynamics	Eidos	82		24	80	0	
June 2006	Titan Quest	Iron Lore Entertainment	THQ	77		30	39	0	533063
July 2006	Prey	Human Head Studios	Take Two	81		62	30	0	750000
September 2006	Defcon	Introversion Software	Introversion Software / Valve	84	$ 95.000	12	2	0	100000
November 2006	Tony Hawk's Downhill Jam	Toys for Bob	Activision	69		18	40	0	924721
November 2006	Resistance: Fall of Man	Insomniac Games	Sony	88		24	100	0	

17 Bibliography

Agile Alliance. (2001). *Manifesto for Agile Software Development*. Retrieved February 25, 2007, from http:/ / agilemanifesto.org/

Alden, S. (2000, December 20). *Postmortem: Ritual Entertainment's Heavy Metal: F.A.K.K. 2*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20001220/ alden_01.htm

Arey, D. (2004, January). Naughty Dog's Jak II. *Game Developer* , pp. 40-48.

Ballard, G. (2000). Positive vs. Negative Iterations in Design. *Proceedings Eighth Annual Conference of the International Group for Lean Construction*. Brighton, UK.

Barrett, K., Harley, J., Hilmer, R., Posner, D., Snyder, G., & Wu, D. (2002, February 27). *Postmortem: Pseudo Interactive's Cel Damage*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20020227/ wu_01.htm

Beck, K. (2002). *Introduktion til Extreme Programming*. IDG.

Beck, K., & Andres, C. (2005). *Extreme Programming Explained - Embrace Change - Second edition*. Addison Wesley.

Bernstein, R. (2002, January 22). *Postmortem: Frog City's Trade Empires*. Retrieved February 25, 2007, from Gamaustra.com: http:/ / www.gamasutra.com/ features/ 20020125/ bernstein_01.htm

Biessman, E., & Johnson, R. (2000). *Postmortem: Raven Software's Soldier of Fortune*. Retrieved December 14, 2006, from Gamaustra.com: http:/ / www.gamasutra.com/ features/ 20000927/ biessman_01.htm

Bilas, S. (2000). *Postmortem: Sierra Studios' Gabriel Knight 3*. Retrieved December 14, 2006, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20001011/ bilas_01.htm

Blossom, J., & Michaud, C. (1999, August 13). *Postmortem: LucasLearning's StarWars DroidWorks*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19990813/ droidworks_01.htm

Bono, E. d. (2000). *Six Thinking Hats*. Penguin Books Ltd.

Bradshaw, L. (2005, January). Avoiding Sequelitis in the Sims 2. *Game Developer* , pp. 38-43, 63.

Brooks, F. (1975). *The Mythical Man-Month - Essays on Software Engineering.* Addison Wesley.

Cage, D. (2006, June). Indigo Prophecy - The nightmare of the original concept. *Game Developer* , pp. 24-29.

Carter, B. (2003, April). Lost Toys' Battle Engine Aquila. *Game Developers* , pp. 50-58.

Cerny, M., & John, M. (2002, June). Game Development - Myth vs. Method. *Game Developer* , pp. 32-36.

Chan, K., Spagnolo, S., Stevens, S., Hagger, N., Chau, D., & Carlton, G. (2003, March 17). *Postmortem: Blue Tongue Software's Jurassic Park: Operation Genesis.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20030317/ chan_01.shtml

Chaveleh, A. (2004, November). Two Timing: Timegate Studios' Kohan II and Axis & Allies. *Game Developer* , pp. 34-38.

Chey, J. (2002, May). Irrational Games' Freedom Force. *Game Developer* , pp. 44-49.

Chey, J. (1999, December 9). *Postmortem: Irrational Games' System Shock 2.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991207/ chey_01.htm

Christensen, S., & Kreiner, K. (1991). *Projektledelse i løst koblede system - ledelse og læring i en ufuldkommen verden.* Jurist- og Økonomforbundets Forlag.

chromatic. (2003). *Extreme Programming - Pocket Guide.* O'Reilly Media.

Condon, R. (2002, October). Z-Axis's Aggressive Inline. *Game Developer* , pp. 42-48.

Cooper, R. (2006, August). Reestablishing an Icon - The peak and pitfalls of Tomb Raider: Legend. *Game Developer* , pp. 24-28.

Corry, C. (2001, August 1). *Postmortem: Lucas Arts' Star Wars Starfighter.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20010801/ corry_01.htm

Darsø, L. (2005). *Findes der en formel for Innovation?* Børsens Ledelseshåndbøger.

Darsø, L. (2001). *Innovation in the Making.* Samfundslitteratur.

Darsø, L. (2005). *Prejekt frem for projekt*. Retrieved December 16, 2006, from http:/ / www.itu.dk/ courses/ I/ F2006/ session10.pdf

Delay, C., Arundel, V., Arundel, T., Chambers, G., & Knottenbelt, J. (2006, December). Detonating Introversion's DEFCON. *Game Developer* , pp. 26-29.

Denman, S. (2000, April 18). *Postmortem: Surreal Software's Drakan: Order of the Flame*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000418/ denman_01.htm

Engel, T. (2002, May 1). *Postmortem: Factor 5's Star Wars Rogue Leader: Rogue Squadron II*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20020501/ engel_01.htm

Esmurdoc, C. (2005, August). Head Games - Double fine's Phychonautic break. *Game Developer* , pp. 30-38.

Fischer, I., & Street, G. (2003, Febuary). Developing Sequels: The Designer's Dilemma - Ensemble Studios' Age of Mythology. *Game Developer* , pp. 44-52.

Freedman, D. H. (2000). *Corps Business - The 30 Management Principles of the U.S. Marines.* HarperCollins.

Frior, M. (2002, February 13). *Postmortem: Mythic's Dark Age of Camelot*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20020213/ firor_01.htm

Fristrom, J. (2002, August). Postmorte: Treyarch's Spider-Man. *Game Developer* , pp. 48-55.

Fristrom, J. (2004, September). The Swinging System of Treyarch's Spider-Man 2 Game. *Game Developer* , pp. 26-34.

Fullerton, T., Swain, C., & Hoffman, S. (2004). *Game Design Workshop - Designing, Prototype and Playtesting Games.* CMP Books.

Fulp, T., & Baez, J. (2005, May). Indie Power - Riding the FBI with Alien Homainid. *Game Developer* , pp. 28-35.

Gabler, K., Gray, K., Kucic, M., & Shodhan, S. (2006, October 26). *How to Prototype a Game in Under 7 Days: Tips and Tricks from 4 Grad Students Who Made Over 50 Games in 1 Semester*. Retrieved January 23, 2007, from Gamasutra.com:
http:/ / www.gamasutra.com/ features/ 20051026/ gabler_01.shtml

Gold, J. (2004). *Object-oriented Game Development*. Addision Wesley.

Goodsill, J. (2006, October). Iron Lore's Titan Quest. *Game Developer* , pp. 34-42.

Gray, M. W. (2004, February). Totally Games' Secret Weapons over Normandy. *Game Developer* , pp. 40-45.

Greig, S., Muzyka, R., & Ohlen, J. (2002, November). Bioware's Neverwinter Nights. *Game Developer* , pp. 42-48.

Hao, W. D. (2003, July 13). *Postmortem: Tom Clancy's Splinter Cell*. Retrieved February 25, 2007, from Gamasutra.com:
http:/ / www.gamasutra.com/ resource_guide/ 20030714/ hao_01.shtml

Hastings, B. (2005, February). Up Your Arsenal: On and Offline in Ratchet & Clank. *Game Developer* , pp. 28-32, 41-43.

Hirabayashi, Y. (2005, October). The Graphical Styling of Resident Evil 4. *Game Developer* , pp. 26-33.

Hubbard, C. (2003, January). Monolith's No One Lives Forever 2: A spy in H.A.R.M.'s Way. *Game Developer* , pp. 48-55.

Hubbard, C. (2001, June 8). *Postmortem: Monolith's No One Lives Forever*. Retrieved February 25, 2007, from Gamasutra.com:
http:/ / www.gamasutra.com/ features/ 20010608/ hubbard_01.htm

Hudson, C., Musyka, R., Ohlen, J., & Zeschuk, G. (2003, December). Combat System Development on Bioware's Star Wars: Knights of the Old Republic. *Game Developer* , pp. 42-46.

Huebner, R. (2000, August 2). *Postmortem of Nihilistic Software's Vampire: The Masquerade -- Redemption*. Retrieved February 25, 2007, from Gamasutra.com:
http:/ / www.gamasutra.com/ features/ 20000802/ huebner_01.htm

Hunicke, R., LeBlanc, M., & Zubek, R. (2001-2004). MDA: A Formal Approach to Game Design and Game Research. *Game Developers Conference*. CMP Books.

Hunt, A., & Thomas, D. (2000). *The Pragmatic Programmer.* Addison Wesley.

IGDA. (2004). *Quality of Life White Paper.* IDGA.

Imamura, A., & Yamoka, A. (2005, March). What's Inside the Room? The Horror of Silent Hill 4 Investigated. *Game Developer* , pp. 34-40.

Imislund, C. (1999, December 8). *Postmortem: Activision's Heavy Gear 2.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991208/ imislund_01.htm

Imlash, W. (2001, October 26). *Postmortem: Startopia.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20011027/ imlach_01.htm

Jacobson, B., & Speyer, D. (2005, November). Scaling the Cabal - Valve's design process for creating Half-Life 2. *Game Developer* , pp. 20-28.

Jobling, P. (2003, December 24). *Postmortem: Eutechnyx' Big Mutha Truckers.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20031224/ jobling_01.shtml

Kijanka, B. (2002, September). Gas Powered Games' Dungeon Siege. *Game Developer* , pp. 42-49.

Larman, C. (2004). *Agile & Iterative Development - A Manager's Guide.* Addison Wesley.

Leighton, J., & Derrick, C. (1999, October 8). *Outrage's Descent 3.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991008/ descent_01.htm

Lemarchand, R. (2006, February). From smart to finish - Jak X: Combat racing and the Naughty Dog production method. *Game Developer* , pp. 17-22.

Leonard, T. (1999, July 9). *Postmortem: Looking Glass's Thief: The Dark Project.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19990709/ thief_01.htm

Li, J. (2002, August 15). *Postmortem: Pixelogic's The Italian Job.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20020815/ li_01.htm

Long, M. (2004, August). The cinematic effect of Zombie Studios' Shadow Ops: Red Mercury. *Game Developer* , pp. 34-39.

Lopiccolo, I., & Rigopulos, A. (2003, August). Harmonix's Amplitude - The Sound of the Fury. *Game Developer* , pp. 40-45.

Löwgren, J., & Stolterman, E. (1998). *Design av informationsteknik.* Samfundslitteratur.

Malenfant, D. (2000, January 7). *Postmortem: Shiny Entertainment's Wild 9.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000107/ wild9_01.htm

Mallat, Y. (2004, April). Ubisoft's Prince of Persia: The Sands of Time. *Game Developer* , pp. 46-51.

Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., & Stage, J. (2001). *Objekt Orientering Analyse & Design*. Marko.

McConnell, S. (1996). *Rapid Development - Taming Wild Software Schedules.* Microsoft Press.

Meynink, T. (2000, July 28). *Postmortem: Angel Studios' Resident Evil 2 for the N64.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000728/ meynink_01.htm

Millinger, M. (2002, June). 2015's Medal of Honor: Allied Assault. *Game Developer* , pp. 50-58.

Min, A. (2000, January 5). *Postmortem: Multitude's Fireteam.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000105/ fireteam_01.htm

Molyneux, P. (2001, June 13). *Postmortem: Lionhead Studios' Black & White.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20010613/ molyneux_01.htm

Napier, J. (2000, March 6). *Postmortem: Sierra's SWAT3: Close Quarter Battle.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000306/ napier_01.htm

Norman, D. A. (2002). *The design of everyday things.* Basic Books.

Oakden, T. (2001, April 20). *Postmortem: Micro Forte' s Fallout Tactics.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20010420/ oakden_01.htm

Passetto, C. (1998, December 4). *Postmortem: DreamForge's Sanitarium*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19981204/ pasetto_01.htm

Pease, S., & Findley, C. (2006, March). Gun - The Good, The Bad and The Ugly. *Game Developer* , pp. 34-43.

Pelletier, B., Gummelt, M., & Monroe, J. (2001, February 7). *Postmortem: Raven Software's Star Trek: Voyager—Elite Force*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20010207/ pellertier_01.htm

Poix, X. (2006, April). Ubisoft's Peter Jackson's King Kong. *Game Developer* , pp. 28-36.

Poppendieck, M., & Poppendieck, T. (2007). *Implementing Lean Software Development - From Concept to Cash*. Addison Wesley.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development - An Agile Toolkit*. Addison Wesley.

Pratchett, T. (2001). *The Last Hero*. Victor Gollancz Ltd.

Price, T. (2003). *Postmortem: Insomniac Games' Ratchet & Clank*. Retrieved December 14, 2006, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20030613/ price_01.shtml

Pritchard, M. (2000, March 7). *Postmortem: Ensemble Studios' Age of Empires II: The Age of Kings*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000307/ pritchard_01.htm

Reinhart, B. (2000, June 9). *Postmortem: Epic Games' Unreal Tournament*. Retrieved January 19, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000609/ reinhart_01.htm

Rhinehart, C. (2006, November). Creeping Dead - Designing the Deathwalk System in 3D Realms' and Human Head Studios' Prey. *Game Developer* , pp. 30-36.

Ridgway, W. (2000, February 1). *Postmortem: Zombie's SpecOps: Rangers Lead the Way*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000201/ ridgeway_01.htm

Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing.

Rookie, F. (2003, October). Monolith's Tron 2.0. *Game Developer* , pp. 42-47.

Rosenkranz, K. (2003, March). Piranha Bytes' Gothic II: The Interactive Score. *Game Developer* , pp. 54-58.

Rouse III, R. (2002, July). Surreal Software's Drakan: The Ancients' Gates. *Game Developer* , pp. 40-46.

Rouse III, R. (2004, May). The Game Design of Surreal's The Suffering. *Game Developer* , pp. 36-42.

Royce, W. W. (1970). Managinge the Development of Large Software Systems. *Proceedings of IEEE Westcon* , 328-338.

Ryan, T. (1999, October 19). *The Anatomy of a Design Document, Part 1: Documentation Guidelines for the Game Concept and Proposal.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991019/ ryan_01.htm

Ryan, T. (1999, December 17). *The Anatomy of a Design Document, Part 2: Documentation Guidelines for the Functional and Technical Specifications.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991217/ ryan_01.htm

Saladino, M. (1999, November 19). *Postmortem: Sierra's SWAT3: Close Quarters Battle.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991119/ startrekpostmortem_01.htm

Saunders, K. (2005, April). Jedi Mind Tricks - Choice and Consequence in Star Wars: Knights of the Old Republic 2. *Game Developer* , pp. 30-36.

Schadt, T. (2007, January). Postmortem: Not Your Typical Grind - Tony Hawk's Downhill Jam for Wii. *Game Developer* , pp. 30-37.

Schaefer, E. (2000, October 25). *Postmortem: Blizzard Entertainment's Diablo II.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20001025/ schaefer_01.htm

Schultz, C. P., Bryant, R., & Langdell, T. (2005). *Game Testing All in One.* Thomson Course Technology PTR.

Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum.* Prentice Hall.

Seropian, A. (2006, January). Chomping at the Bit - Wideload Games' Studio Experiment Bites Back with Stubbs the Zombie. *Game Developer* , pp. 24-31.

Sheffield, B. (2006, April). The Wright Stuff - Will Wright on life, the universe, and everything. *Game Developer* , pp. B1-B9.

Simpson, J. (1999, May 21). *Postmortem: Raven Software's Heretic II.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19990521/ heretic_01.htm

Smith, B. (2001, October 10). *Postmortem: Poptop Software's Tropico.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20011010/ smith_01.htm

Smith, M. (2007, February). Postmortem: Resistance: Fall of Man. *Game Developer* , pp. 28-36.

Sobek II, D. K., Ward, A. C., & Liker, J. K. (1999, Winter). Toyota's Principles of Set-Based Concurrent Engineering. *Sloan Management Review* , pp. 67-83.

Sommerville, I. (2001). *Software Engineering - 6th Edition.* Addison Wesley Publishing.

Sontag, S., Drew, C., & Drew, A. L. (2000). *Blind Man's Bluff.* Arrow.

Spanel, O., & Spanel, M. (2001, December 19). *Postmortem: Bohemia Interactive Studios' Operation Flashpoint.* Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20011219/ spanel_01.htm

Spector, W. (2000, December 6). *Postmortem: Ion Storm's Deus Ex.* Retrieved Feburary 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20001206/ spector_01.htm

Stojsavljevic, R. (2000). *Postmortem: Westwood Studios' Command & Conquer: Tiberian Sun.* Retrieved December 14, 2006, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000404/ tiberiansun_01.htm

Sussman, D. (2006, February). The Buzz of Harmonix's Guitar Hero. *Game Developer* , pp. 24-28.

Takahashi, K. (2004, December). The Singular Desing of Katamari Damacy. *Game Developer* , pp. 34-38.

Thomas, G., Morichere-Matte, S., & Mosqueira, J. (2003, November). Developing a Sequal: Evolution, Not Revolution - Relic Entertainment's Home World 2. *Game Developer* , pp. 40-45.

Train, T., & Reynolds, B. (2003, June 27). *Postmortem: Big Huge Games' Rise of Nations*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20030627/ train_01.shtml

Upton, B. (2000, January 21). *Postmortem: Redstorm's Rainbow Six*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20000121/ upton_01.htm

VandenBerghe, J. (1999, December 3). *Postmortem: The X-Files*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19991203/ xfiles_postmortem_01.htm

Ward, A., Liker, J. K., Cristiano, J. J., & Sobek II, D. K. (1995, Spring). The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster. *Sloan Management Review* , pp. 43-61.

Weinberger, D. (2002). *Small Things Loosely Joined*. Perseus Publishing.

White, S. (2002, July 10). *Postmortem: Naughty Dog's Jak and Daxter: the Precursor Legacy*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 20020710/ white_01.htm

Wikipedia.org. (2007, January 24). *Toyota*. Retrieved January 24, 2007, from Wikipedia.org: http:/ / en.wikipedia.org/ wiki/ Toyota

Wyckoff, R. (1999, May 14). *Postmortem: Dreamworks Interactiv's Trespasser*. Retrieved February 25, 2007, from Gamasutra.com: http:/ / www.gamasutra.com/ features/ 19990514/ trespasser_01.htm

Young, G., Rodriguez, M., & Pickford, C. (2004, March). Bizarre Creations' Project Gotham Racing 2. *Game Developer* , pp. 46-54.