University of Bergen

Department of linguistic, literary and aesthetic studies

DIKULT350

Master's Thesis in Digital Culture

Fall 2020

Software as an art

The aesthetic influence in software development

Jostein Enes

Abstract Norwegian

Denne master oppgava bruker litteratur og intervjuer til å se nærmere på hvordan estetikk og system utvikling hengersammen og påvirker hverandre. Oppgava tar utgangspunkt i Warren Sack sin bok "The Software Arts". Sack argumentere i denne boka at de liberal arts er kjerna i databehandling. Måten Sack argumentere for dette er ved å se på historie og språk (programmering). Denne oppgava argumentere at de liberal arts er et for stort tema til å kunne argumentere imot og at man må se nærmere på hver enkel bit innad i det. Derfor omhandler denne oppgava estetikk i systemutvikling, systemhåndtering, programmering og programvare design.

Målet er å finne ut hva de som jobber med dette tenker og føler for å kunne se hvordan de påvirker produktet som blir lagd. Oppgava gir også overblikk over hvordan dette har endret seg i tritt med samfunnet, fra et produktfokus mot et brukerfokus, på rundt 2000 tallet.

Oppgava viser til at det finnes noe estetisk vakkert med systemutvikling og emnene funnet i det. Det blir også argumentert for at det er mer til systemutvikling enn programmering, noe som gjør at andre utdanninger som ikke er innenfor data har en plass i utviklingen av programvare.

Metodene som oppgava brukte, gjorde at argumenter kunne begrunnes seg i forsker artikler med kommentarer ifra utviklere rundt hva de mente er estetisk med systemutvikling. Funnene som ble gjort viser at det er estetiske attributter med systemutvikling. Ett eksempel er utvikleres positive følelser rundt å lage et produkt som brukarene trenger og får nytte av.

Abstract English

This thesis uses literature and quantitative interviews to look closer at how aesthetics and software development is connected. The thesis springs of from Warren Sacks claim, in “The Software Arts”, that at the centre of computing is the liberal arts. In this book Sack only focused on language and programming something that this thesis found lacking. Since aesthetics is a large part of humanities and the liberal arts, it can therefore be argued that aesthetics is also a part of the centre of computing.

Because of this this thesis is investigating not just at programming but software management and software design as well, to see where aesthetics can be seen and how it has affected software development. The thesis therefore gives definition and explanation to what aesthetics is in the three topics just mentioned, programming, software management and software design. Before using these definitions to create a fourth definition around the aesthetics of software development.

The thesis is trying to show the aesthetic beauty of software development and argues that there are more to software development then coding and mathematics. It also takes a closer look at outside forces that has helped change what developers have found aesthetic through the last few decades.

The method this thesis used allowed the arguments to build on scientific articles and check these up towards what developers in businesses thought about aesthetics. The findings were that the developers in the businesses showed a great interest in some aesthetic attributes, specifically working to create a good product for the user gave them positive feelings.

Acknowledgments

Before starting this thesis, I would like to give my thanks to Daniel Jung, my advisor who helped me create the framework of this thesis. As well as Nicholas Montfort who gave me advises in the coding chapter of this thesis. Without them this thesis would have been a lot harder to write then what it was.

I would also like to give thanks to the person that has been my mental note board, my mother. She has been there and had to survive all my ramblings about aesthetics and software development, and it is thanks to her that I've been able to see the larger change in society and finish this thesis with my mind almost intact.

There are also my fellow students at Digital Culture. Thanks for the student meetings we have had throughout the semester, it has been good to talk and see some faces in the pandemic and know that we are all in this boat together.

I would like to thank my better half. She might be disagreeing that she has helped with this thesis, but the fact that she is there for me no matter what pushes me to do better.

Lastly, a thank to you, reader of this thesis. I hope that my thoughts, ramblings and findings might be of use to further studies on culture, aesthetics and software development.

Table of Content

Introduction

This thesis topic is the aesthetics in software development and the research question is “**How has software development changed since it was introduced and in what way has aesthetics been a part of that change?”**

There are many reasons for this topic, but one of the main once is Warren Sacks book “The Software Arts”. In this book he brings up that at the root of programming is the liberal arts. This thesis argues that this claim is too broad of a statement and nobody could argue against that. Instead, there are aspects of the liberal arts that has had a stronger focus then others. Mathematics and science have been the cornerstones of software development and programming. What Sack is really trying to bring up is the discussion that humanities, philosophy and the fine arts has a part here as well. The only part Sack looks at is programming.

This thesis is therefore focusing away from a purely programmer centric viewpoint, and rather towards a software development one. There is more to software then just programming. If one is to look at the “Arts” in software development, then it is important to have an overview of it all. Because of the size of the thesis topic, the specific topic that will be focused on is the aesthetic of software development. This is a topic Sack did not touch on, but it would have brought much to his investigation of programming as a literature.

Another topic that inspired this thesis is the cultural change that hit software development in the early 2000’s. This thesis will look closer at the changes that happened, why this change happened and how developer’s aesthetical judgment has changed because of this. Few studies and thesis are looking at how the focus change has affected the developers, but rather on what benefits one approach has over the other. This thesis on the other hand, tries to see the developer and what they want to gain from software development.

The thesis topic is closely linked to user experience for the end users as well as for the developers of software. The research that has been done around this has often been focused towards the users of the technology and not on the creators. This is

another reason that this thesis will be focusing primarily on the creators and their view and thoughts on what they're doing.

The way this will be done is by presenting what aesthetics are and how it changes based on what topic is being looked at and who you ask. Then it will analyse the qualitative interviews that have been done with developers and cross examining these against the literature that has been found. Part one starts with looking at the definitions that needs to be explained, these are aesthetics, software development and liberal arts. From there it gives a rundown of the methods used to gather information, literature search and quantitative interviews. Before lastly giving a summary of "The Software Arts".

In part 2 it looks closer at the aesthetics found in software development from two viewpoints in four different parts of system development. Coding, User experience, User interaction and Software management. One of the viewpoints is from Warren Sack who views coding as closely related to literature, creativity and the "liberal arts". While the other point of view sees a more systematic description over how the systems and machine works. This viewpoint believes that system development is the same as creating any machine and needs a robust framework. We will also look at how these views have changed since the introduction of the agile way of thought and try to explain why these changes happened.

Specifically, for coding, we will be looking at how these viewpoints have changed how coding is being done. As well as looking closer at how aesthetics has become a tool in creating code, investigating code and reading code. We will also be analysing Warren Sacks claims that coding and literature is closely related and see what the interviewed programmers think about this.

User experience brings up the topic of how aesthetics helps with creating what we call "good" experiences from software development. When you hear that most often a picture of the graphics of the software or models comes to mind. What this thesis will be arguing is that user experience and aesthetics is so much more than that. It also embraces understandability of code, communication between different parts of development, adaptability and creativity. Something that will be looked at closer in the software design chapter.

User interaction on the other hand is, like the name suggests, about interaction. The chapter will be investigating how interacting with code, people and design can be aesthetic. Where user experience looks at the feeling you get from these topics, user interaction is about how the software works and interacts with its users. Is the user understanding the design without problems? Is the way the functions and code works done in a way that allows you to quickly learn what it is doing? And so on. Because the user experience and user interaction topics are so closely related, these two topics is part of the software design chapter.

In Software management will be looking at the methods that have been used and are being used for creating software. It will go deeper into the methods that agile development has put forth and see how it creates a framework that allows for aesthetic thinking and creation. We will also investigate what the project owner finds aesthetic and what their thoughts are on the topics that were mentioned to the developers and designers. How is aesthetics relevant for project owners and what is their view on aesthetical topics for developers? Is one of the questions we will be trying to find out. The reason that this is important to look at is that there has been a conflict between management and developers before. The agile methodology was the reaction to the power imbalance between management and developers. By looking at the aesthetical differences, one can find out if this is still the case.

Lastly, there had to be some cuts and though calls on what should be part of the thesis. Since the topic of aesthetics could be looked at from every single aspect of software development and how it all fits together. The call was made too only focus on the areas the interview objects mentioned and found the most important. This is based on the questions asked and what they wanted to talk about. Therefore, the focus the thesis became the four topics mentioned earlier, coding, user experience, user interaction and software management.

Background

Before going into the thesis, I will give an introduction of where my viewpoint comes from. The reason for this is that I have a bias from my education in system development, something that might affect some of the interview questions that were asked and what sources that were used. To mitigate this, I have been sending my questions to the thesis advisor and used friends and family as sparring partners to make it as unbiased as possible.

From the former mentioned education in system development I started the master program in Digital Culture with a technical perspective. This perspective has changed somewhat. As a developer you do not need to focus on the aesthetics of what you're doing, but as I now can see, it might be useful to understand its effects so one can improve the work and culture surrounding development.

Another important fact I feel should be mentioned, is my passion for user experience and user interaction. These are topics that I care deeply about and has been a driving factor for me to investigate how aesthetics is being used by developers and designers.

1. Definitions and methods

Part one of this thesis will be focusing on giving the reader an overview over the methods and definitions that is needed to be able to follow along with this thesis. This will be done by defining how this thesis will be using specific word and topics like software development and aesthetic. Before explaining the method used to find the information that will be used in part two.

1.1 Definitions

This chapter gives the definitions and how four different words and topic will be used. First it explains what aesthetics will mean in this thesis, then it will be looking at liberal arts. This is because of how often it gets mentioned in Sacks book. Thirdly Software development, what is software development? Fourthly it explains what this thesis means with the word software design.

1.1.1 Aesthetic definition

When someone mentions aesthetics, it is often associated with the fine arts, in form of pictures, painting, sculptures and music. But what is aesthetics really? Immanuel Kant tackles this problem by describing aesthetics as the judgment of taste. He continues saying that judgment of taste has two sides, subjectivity and universality. By subjectivity it means the aesthetic is based on feelings of pleasure and displeasure found within. While aesthetics based on universality is told to us through culture and people. (Zangwill 2019)

Aesthetics in software development is not something that has been described in great deals by other authors. There is tough a field of study that is closely related that has been researching this and that is the field of video games. One of the researchers in the field Kristine Jørgensen, who is on the project “Games, Transgression and Aesthetics”, she defined aesthetics in games around not only the audio visuals but also that gameplay is an important part of the beauty and aesthetics of video games.

> *“The project assumes that games are experienced as aesthetic objects, and that the “aesthetic” or “beauty” can be found in gameplay as well as in content or in the audiovisual features of the game. This means that people playing a game for the sake of good gameplay and interesting challenges also have an aesthetic experience, in a similar way as those appreciating the message or the audiovisual art.”* (Jørgensen 2015).

Jørgensen points out that the enjoyment or beauty of video games comes from not only the visual objects you can see, but also overcoming challenges, the gameplay and music. This is a new way to look at aesthetics and differ from the established views that most people have on the topic. This way of looking at aesthetics started to spread in the scientific and research world after a lecture by Charles Percy Snow named “The Two Cultures and the scientific revolution” (1959) (Meisenberg 2018). Here he brought up the divide between scientific and literary scholarships. One of the things he brought up in this lecture is the split between literary intellectuals (which are art, philosophy and humanities) and other types of science like mathematics and physics. To quote Snow “The intellectual life of the whole of Western society is increasingly being split into two groups, … literary intellectuals at one pole --- at the other scientist … Between the two a gulf of incomprehension.” (Meisenberg 2018) Snow was on the side of science in the debate but with this lecture he defined the divide between the two factions, something both sides agreed on. This divide is something one can notice in the amount of interdisciplinary research that has been done between the two sides.

Where scientist and engineers focus on close frames and product development, game development, on the other hand, has a culture focused around creativity. To develop interesting games, one needs to focus more on the end users then what product developers do. Often games are made to be enjoyed or to send a message. It is therefore important that the person playing the game can consume the product. Which is why game developers and the indie development scene has always had a closer tie to humanities and aesthetics, something that reflects in the research done on the areas.

There are some scholars who think there is a connection between aesthetics and science. One of these scholars is Tauber. He brings up the point that scientists use empirical criteria when figuring out if a theory is true, but this is just one of the principles that are being used.

> *“However, in constructing their notion of what makes a theory acceptable, many scientists refer to concerns other than for the empirical performance of theories. Some of these concerns are aesthetic. Many scientists have possessed a concept of the beauty of theories; they have subjected to*

> *aesthetic appraisal the intellectual constructs that make up theories, and the verdicts of these appraisals have contributed to determining whether they deemed each theory acceptable"* (Tauber 1996)

The point Tauber is trying to get across is that science and aesthetics does not have a gulf between them. This thesis argues therefore that aesthetics and beauty can be found anywhere and ignoring that it exists in empirical theories shows an arrogance and close mindedness to other perspectives. This way of thought has gradually started to change but there is still a public opinion that science and aesthetics does not go hand in hand.

In a study in 2011 called "The Aesthetics of Software Code: A Quantitative Exploration" (Kozbelt, Dolese and Soidel 2012) they investigate if aesthetics can be found in software coding. Their literature search shows that there is a connection but continues with:

> *"Even so, while research linking science and aesthetics continues to proliferate, the focus of research and discussion tends to be mainly in nonapplied highly theoretical domains. In contrast, in more applied or technical domains like software development, investigation of aesthetics are rare, and studies with a quantitative emphasis are virtually nonexistent"* (Kozbelt, Dolese and Soidel 2012)

Kozbelt and his colleagues did follow this up with quantitative interviews of software developers. Where they found out that experts in the field of software development and programming, often find coding and creative artefacts to have an aesthetic appeal. Further in the study it is shown that aesthetics can be used for quality assurance and testing, this is something that will be looked at closer in part two of the thesis.

Another look at this topic was done by Warren Sack in the book "The Software Arts", where he and the other authors on the project investigate different topics of software from a humanistic perspective. In the autumn of 2019, Warren Sack also had a lecture at the University of Bergen, where he introduced the project and claims the book takes. The lecture started with Sack highlighting some part of the book on a

PowerPoint. The part he focused on was Adam Smith's workshops in the 1800's. Sack brings forth that modern computing and digitalization follows the same principles of the workshop made by Adam Smith. Step by step commands that the workers are to follow. These commands were blueprints of different machines with both text and drawings, something that puts engineering and art in a coalition. It wasn't the only part that can be linked to computing. The workshops also brought up specialisation and the division of labour. It was this division and specialisation that made Herbert Simon say that Smith was the inventor of the digital computer. With the division of labour, we got a new way to organize and structure workforce. Herbert Simon saw that this way of dividing tasks is the same way that a computers function. (Sack 2019)

The reasoning for bringing this up, is to argue that the specialisation and division of labour created pride and an intellectual fulfilment. Knowing that one had mastered a part of the larger machine, and with this a feeling of fulfilment from the work that is being done. Ergo, not only the drawings of the blueprint could be aesthetic but with knowledge of the production line, every part of it brings a feeling of enjoyment, with other words aesthetic qualities. The same can be said for the evolution of the medium, software development.

Noël Carrol wrote in his article "the specificity of media".

> *"Use determines what aspects of the medium are relevant for aesthetics, rather than some essential trait of the medium determining the proper use of the medium. But if the use of the medium is key, then effects will be evaluated in terms of how well they serve presiding purposes."* (Carroll 1985)

Aesthetics in software development does have hints of what Noël brings up. The term "spaghetti code" is used by most software developers to point out overly complicated, not to the point, unstructured or difficult to maintain code. Following Noël's thought process, useful, relevant and to the point code will be evaluated as aesthetically pleasing. Something Kozbelt's research into "beautiful vs ugly code" seems to prove. Another of the thing's Noël brings up around this is that one should not use one type of medium to critique another. This could also be said for aesthetics as well. Because of aesthetics being subjective, using painting aesthetics to critique software development or game development would create friction. This is why there needs to

be a definition on what aesthetics in software development is, to be able to critique it correctly.

Therefore, this thesis will be borrowing from Kant's description of aesthetics. That it is subjective as well as universal. It will also be linked with what Noël mentioned, with use. Aesthetics for a developer could be boiled down to how systems, code and graphics are being used and thought about. An aesthetic code as mentioned above is quick, functions as wanted, easy to read and understand. Graphics is measured in how quickly a user finds what he is searching for, if everyone can understand what is being shown and is it good to look at. The point is that aesthetics in software development differ from who is looking at it, but the fact is that good aesthetics brings a measurable benefit, follows throughout the field.

However, this is not the only aesthetic present in software development. This thesis argues that there are many different forms of aesthetics even in software development. The three most normal ones are emotional, practical and visual aesthetics. With emotional aesthetics for software development it means the drive, feeling of achievement and excitement over a topic. These types of aesthetics are harder to measure but can be seen, and in some cases, felt when using the finished product. A well-polished bug free product gives off the feeling that the creators cared about it, something that makes it stand out from other products. Practical aesthetics on the other hand looks closer at the solutions that were implemented. It is here that function, and proficiencies show its face. The things that fall under it are smart design, code quality, UML models and functionality. And lastly visual aesthetics which is the composition of both code and design.

The visual aspect is often what gets contributed to aesthetics because of the similarity to art. In a way, visual aesthetics for software development falls between painting and writing since it encapsulates both. How the code is written as well as the design of the graphical user interface (GUI). These three aspects are influenced by aesthetic emotions and draw from that field of study. More specifically from the reasearch article "What are aesthetic emotions" by Menninhaus and others. (Menninghaus, et al. 2018)

Menninhaus's article puts aesthetics down to four points. One, "full-blown discrete emotions, for all their differences in affective nature, relevant appraisals, and other

emotion components, always include an aesthetics evaluation/appraisal of the object or event under consideration" (Menninghaus, et al. 2018). A example of what this means for the software would be looking at a piece of code or design and get a feeling inside oneself. The second part of aesthetics emotions is that it is differentially tuned to a type of aesthetic virtue. By this they mean that it brings up a specific emotion. Thirdly, its associated with subjectively felt pleasure or displeasure during an emotional episode. With other words, the code or design can bring both positive or negative emotions based on its subjective aesthetics. And lastly, "aesthetics emotion are an important (though certainly not the only) predictor of resultant liking or disliking." (Menninghaus, et al. 2018) The point is that it is through aesthetic emotions that one can measure the effect of the aesthetics of coding and design in software development.

1.1.2 Liberal Arts

Warren Sack uses the term, the liberal arts, as he makes his case for it being in the centre of the computer revolution. (Sack 2019) What does this mean precisely?

Sack never gives a concreate answer to what the liberal arts are, but he mentions a bit about how he uses it

> "The Software Arts is also a reading of the texts of computing-code, algorithms, and technical papers- that emphasizes continues between prose and program. Historically, it is possible to say that this position was first sketched out in the seventieth century in proposals to develop artificial, philosophical language that were used to knit together the liberal arts (e.g., logic, grammar, and rhetoric, the liberal arts of language) and the mechanical arts (e.g., those practiced by artisans in workshops producing pins, stockings, locks, guns, and jewellery)." (Sack 2019)

Based on this one can see that it might have something to do with logic, grammar and rhetoric. Following up on this description by Sack one must find out more specifically what is meant by this. Moore posted in the journal of the American Statistic Association a deep dive into the liberal arts and statistics. He categorizes

liberal arts as “flexible and broadly applicable modes of thinking” before he shows there are two sides to the liberal arts. The philosophers like Socrates that are seeking truth and understanding of the world and the orators that there are “known truths and fixed standards of personal and civic virtue” (Moore 2012). With other words the liberal arts are a term to describe western philosophy and education. From a modern view point it is natural and social science, humanities and arts. (Liberal Arts 2020)

This changes the perspective of what Sack is claiming. With other words, the claim is that liberal arts, natural and social science, humanities and the fine art is at the centre of computing and software development. A claim that is hard to disagree with.

1.1.3 Software development

This thesis is about the aesthetics in software development, but what is software development in this thesis? Software development can be easily described as how to develop software, but that does not explain what it is or does. The field of software development is the planning, designing, creation, implementation and maintenance of digital solutions (aka software). It is about creating solutions to fix problems or inconveniences that customer or end users has through software code and design.

There are many aspects to software development, and they have been changing throughout the decades. This thesis will be looking closer at three parts of it, these three parts are software management, coding and software design.

This thesis is also arguing that when using the term developers, it means everyone that is developing software. Meaning programmers, designers, testers and project owners.

1.1.4 Software Design

Software design is the term this thesis will be using to explore user experience and user interaction. As the name suggests, it should encompass everything design related around software, but because of the scope, the topics that will be brought up are. User experience, user interaction and universal design.

1.2 Methods

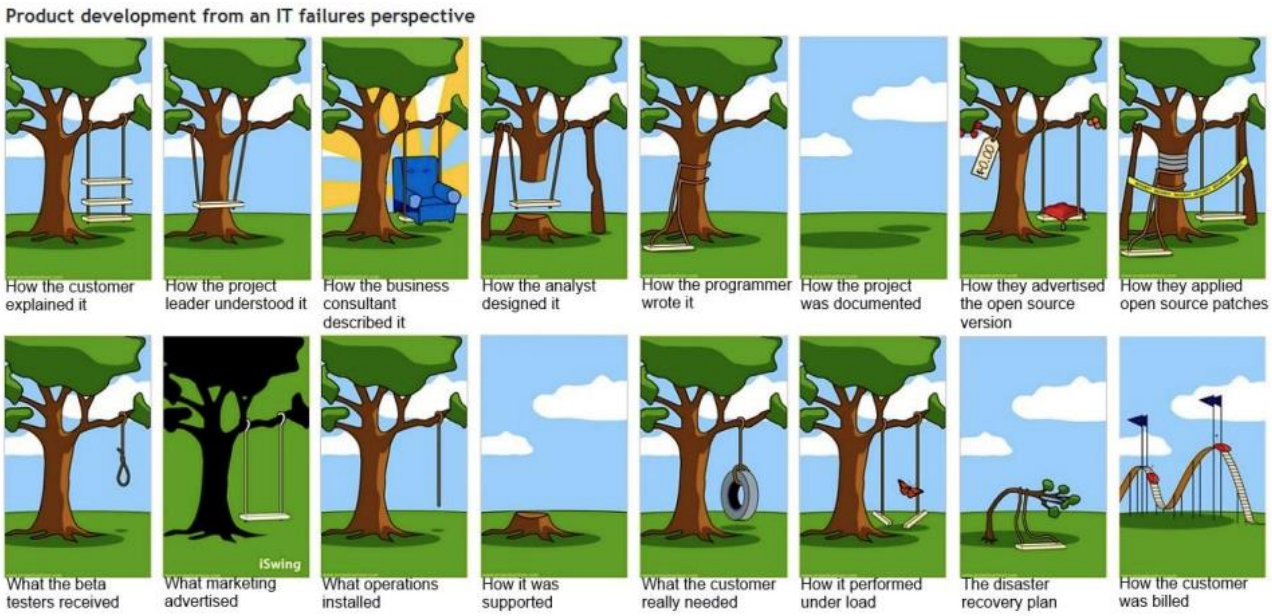
This chapter will be focusing on showing the methods that was used to find the information needed to answer the research question. The way this was done was with the help of interviews and a literature search. The chapter is therefore divided in two, interview and literature.

1.2.1 Interview

With a new understanding of how aesthetics can be interpreted from the viewpoint of humanities and software development, it is important to take a closer look at the research interviews that were done during the thesis. This chapter will not be going over the findings from the interviews but rather the process, difficulties and challenges that presented themselves and how it was dealt with. The findings will be laid forth to build up under arguments or against them in later chapters.

It was decided early that the way this thesis was going to be done was through a dual research method. By this it means both literature and interview research. The plan for the thesis was to focus first on the literature, and then write qualitative interview questions, based on aesthetics and the topics that were being presented in Warren Sacks book. By doing it this way, one could use theory to create questions focused around the topic to increase the chance of getting relevant answers.

While talking about answers. The interview was created to find out what aspects of development different programmers, designers and project owners (management) find aesthetic. Because one cannot mention what one is looking for, the questions that were made followed the comic strip found bellow. This comic strip is a well-known comic, pointing out the different failures often found in software development



(unknown 2020)

This comic strip was introduced to the interview objects with the question: what is it you find most relevant in your daily work based on this comic? Followed by, is there some of these points you think mirrors reality more than others? As an icebreaker the comic did its job and sent the interview towards topics that the interview object was interested in. From there, questions about more generalised topics began. These topics were, user experience, coding, software development and user interaction. There were questions outside of these topics, around security and ethics, that might be brought up if relevant.

The interview objects were put into three categories, designers, programmers and project owners. They all got the same questions except for the branching ones that naturally came up. The hope was that by talking to a wide spread of people who had different roles in software development, it would give more data on the different views on what aesthetics in software development is. Since aesthetics changes based on people, it was important for this study to gather information from people with different backgrounds. This was to gather a better picture of how they interact with aesthetics across different roles.

Before doing the interviews, there was some prejudice against what the findings was going to be found. One of these was that there would be a big difference between project owners and programmers in what they found as interesting topics. Another one, was that programmers would be less aware of aesthetics than designers, this

prejudice is based on their education and the development culture that is being thought in higher education. In the same way the difference between the two businesses was thought to be rather severe before starting, one business is a product company while the other is focused on consulting. These prejudices will be analysed and answered at the end of the chapter to see if where true or not.

As mentioned, this was a qualitative interview that interviewed interested people at two businesses in and around Bergen. The plan was to meet them in person to increase familiarity and get a good dialog around the topics. This came to an end early march because of the pandemic closing most of society down. The solution was to do this over the internet, and with it came a few more challenges. First off, was that the records of the conversations would have to be saved on a computer or server. This meant that one needed a secure server and access to it from and save it. This was done by contacting UIB and getting a SAFE account. SAFE is UIB's way to connect to the local server at the university. Here all the recordings were saved while encrypted with a key that is saved on an external disk. The project was also sent to NSD (the Norwegian centre for research data) for confirmation that it was done after the law.

After all this was done, the interviews were done over a two-week time in May / June. The first interview had some audio issues, where the recording did put interviewers voice and the interviewees voice in two different recordings, and it had to be patched into each other. This left it with some places where they talk over one another. The program Voicemeter Banana was then downloaded which works as a recorder, and both inn and output volume was recorded successfully on the next few interviews.

Overall, the interviews were a success, even though they took longer than first anticipated to complete, something that is to be expected with a global pandemic. The data was then encoded and safely stored on SAFE through anyconnet.

The interviews were created both in English and Norwegian, but since all the interviewees was native Norwegian speakers it was held in Norwegian. This means that all the quotes and thoughts from the interviewees are translated from Norwegian to English in this thesis.

1.2.2 Literature

The literature used in this thesis has been found with the help of google scholar, the University of Bergens (UiB) library search database Oria, and with the help of a librarian at the university library. The book “The Software Arts” was found because of a lecture the author held at UiB in 2019. The last method used to find literature, was going through the sources of articles that was found in said search engines. This was done to check what the original articles or books said on the topic and to find more information and to double check its legitimacy.

Examples of the sources that was searched for are “aesthetics in software development”, “aesthetics in coding”, “software development aesthetics” and “aesthetics in games”. Since the search engines read every word and use them as keywords to find related articles, the focus was on finding related topics that mentioned or talked around what this thesis investigates.

To make sure that the quality of the sources was kept, only research articles and books was used. When it came to books it was also investigated what was written to make sure it made sense based on sources and context.

A problem that came up under the search for sources was that most articles that google scholar found, were behind a paywall. The solution became to use google scholar to find relevant sources and the Oria database to gain access to them, since UiB allows students access to most articles that are pay to view. Because of this, a wide scope of articles and papers was accessible. The share amount of information led to it being categorized by date, and the newest and most cited was investigated first. The reason for this is that they often draw sources from the older papers, which means that by reading the newer ones you would also be able to follow the path backwards, to see the whole picture instead of making it from scratch.

The second problem that was found was that aesthetics in software development is not something that has been defined by anyone. Most often they mention it in passing and just hope that people know what it means. Therefore, this thesis looked deeper into aesthetics by following sources which led nowhere and had to go looking into the study of aesthetics to create an argument for what aesthetics is in software

development instead. The research article that helped the most was “The Aesthetics of software code: A Quantitative Exploration” (Kozbelt, Dolese and Soidel 2012) that looks specifically at coding and not development as a whole and is something that will be looked at in chapter 2.

1.3 The Software Arts

In the book “The Software Arts”, Sack argues that computing and software is a part of the liberal arts, of humanities and literature. He quotes many renowned software developers, like Steve Jobs and Edsger Dijkstra, that share his views. Sack then follows up by critiquing universities and societies view on computing. The view he is criticize is that programming is a part of science or engineering and nothing else. Sack continues to point out how scientists and engineers coined the term “Software engineering” in 1968 to create a link between software development and engineering, even though many disagreed with it then. (Sack 2019)

Only two years later, Winston W. Royce posted an article called “Managing the Development of Large Software Systems”. Here Royce introduced a couple of different approaches and thoughts on how to develop and manage large software systems. Royce did not name the development processes he thought of, but the first method he mentioned was named Waterfall by others. (Royce 1970) An interesting fact is that this is the one methodology that became popular (Waterfall) and is the one Royce thought had the greatest flaw. This methodology was an example on how not to develop. He thought that one needed to use an iterative method, something that is much closer to how development is today.

> *“I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the*

development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs." (Royce 1970)

The method is very closely related to how engineers work on their projects. Which is something that could be used by the software engineering camp, to argue that software development is part of their camp and not humanities and arts. One can argue that because of the similarities, the art and aesthetics side of development was buried by the science field. Universities taught students that one had to be good at math and logic to develop software and that creativity, humanities and art was secondary.

This trend has changed after those that thought programming should have more freedom created the agile manifesto in 2001. (Grenning, et al. 2001) The manifesto contains 12 principles on how programming and development should be done. The manifesto started a cultural change in software development, and today almost all projects follow some of these principles. With this cultural change, software development has changed in the way Sack wanted it too. There is less stigma for looking at development from an art perceptive, but schools are hanging behind in this change.

Sack then comes with three claims: One, education needs to change to mirror the change in the field and integrate liberal arts into the teaching of software development. Two, software can be written in the manner of an artist/humanist, one does not need to be good at math or an engineer to understand how to write software, it is a language on how to do tasks and not a mathematical argument on how the world works. Third, if software is the new lingua franca, then there are a series of ethical and moral questions that must be pursued. And lastly, the most general claim, is that software arts are a new name for something that has been going on for a long time. He then argues that he, just like Steve Jobs, puts arts in the centre of software. (Sack 2019)

This thesis will be investigating Sacks arguments closer in later chapters, but it is important to look a bit closer on the history of the Software Arts. In the 1800's Adam Smith started the first workshops and created the first work language. It describes what artisans, designers and artists are supposed to do. Instead of being a mathematical language, it is a description of how work is to be done. Looking

forward, many great minds have taken inspiration from this work language Smith created. One of them is computer scientist and Nobel prize winning economist Herbert Simon, who said Smith was the inventor of the computer. (Sack 2019)

Based on the information that Sacks puts forth, one can see that there is some proof that literature and humanities is a part of software and computing. The focus of “The Software Arts” is on software and coding, specifically its ties to literature and writing. Where one can see the argument for it, there are also many more aspects to it that could be looked deeper into. One of these aspects is aesthetics and how it fits into software as an art.

The following chapter will be taking a deeper look at different topics in system development, to figure out where aesthetics and the arts mentioned by Sack fits into the overarching field of software development.

2. Deep dive into system development

This part is focusing on the three topics Coding, Software management and Software design. The structure of the three topics will be, an introduction to each topic, some history around the topic, how it has grown and why it is what it is today. The thesis will be highlighting what aesthetics is inside of these topics, and how it is being used to improve culture, standards and the product. There will also be quotes from the interviews. These will be used to argue, for, against or as extra information around the themes of aesthetics, humanities and liberal arts in software development.

2.1 Software management

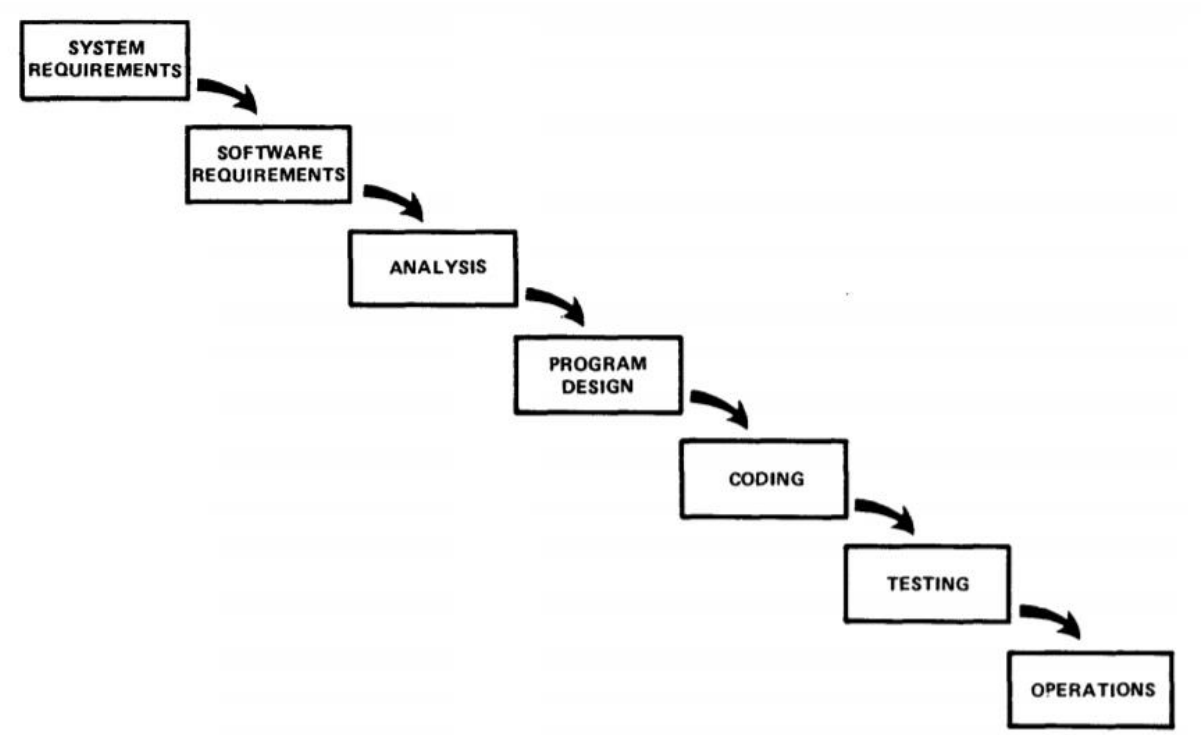
In this chapter we will be looking closer at how software management has changed throughout the years. From a business focused approach, on requirements of the customers, towards the needs of the end-users. It also brings up two different aesthetical beauties found in the different approaches, and the interview findings show how the different parts of software development views these methods. At the end of this chapter, it will be shown how this change happened and what outside forces might have influenced this change.

2.1.1 Traditional methodology

To really understand software development and investigate how aesthetics is being used inside of the development process, one need to understand the two viewpoints that exists in software development. As mentioned earlier, these two viewpoints are the engineering and mathematical view, with their detailed descriptions of how to develop and code. The side is the agile method which comes from a humanities approach, with its focus on creativity and adaptability. These two sides are often called traditional and agile, which is something this thesis will be using from now on.

The traditional method was created from the viewpoint of engineers and mathematicians, as a guideline to make sure software got developed in a sustainable way.

Royce, in his paper around development methodologies, puts fourth different methods to develop software in a safe and efficient way. This came from his personal view on managing large software development projects, mainly from spacecraft missions and post flight-analysis. He continues to point out that a step by step guide will lead to problems, because if all information is not known at the start, one will need to return to start if more information gets uncovered throughout development. (Royce 1970)



(Royce 1970)

Most did not take Royce concerns to heart and started using his model to develop software. A study around traditional and agile development at the university of southern Denmark, described it as *"Traditional (or rich) processes aim to address the whole software project lifecycle, e.g., by providing comprehensive guidelines, standardized procedures, project planning templates, and interfaces to further organization processes"* (Theocharis, et al. 2015). This means that the workload to describe the project, before it started, became massive in bigger projects. If information was uncovered after the start, one could not change it without extra costs. This is something that effected the user experience for the end users the most, because they were the last priority, and the time and money was used up by the time they got to see the product.

This is not to say that this way of development is universialy bad. There are times where one have all the required information and no changes will happen. It is like creating a car, one starts with planning the car, then the components of said car, before analysing and designing it, followed by putting it togheter and testing it before release. In small well defined settings, it works wonders. It is when the goal post keeps moving that this approach to development starts having problems. Another

problem is when projects gets to complicated. If this happens, the planning will take a along time, since it is necessary to be sure before starting production.

The way waterfall works, is the same way as the workshops Waren Sack mentions by Adam Smith. Smiths workshops describes how to create a product, by designing and creating blueprints for every part of the process. The workshops then starts to assemble, testing and lasty implementing the product described. With other words, the aesthetics of this way of development is about structure, planning and security.

Someone that argues that the traditional methods is the best, is the “Software development: Agile vs. Traditional” by Marian, Marinela and Bogdan at the Buchacest University of Economics studies. They argue that having less documentaion, will make it harder for new people joining a project. These people will need to ask more question and by that decreasing the productivity. They also argue that periodic meetings with the client are boring and tiring. This is because having to present the modules repeatedly to the client, to new members and the end users takes away from time that they could have used on coding. Another argument is that short iterations, creates thight schedules, that will take time away from complex parts and making it hard to create those. (Stoica, Mircea and Ghilic-micu 2013)

Which might be true, but the way they are framing the agile method is from the view point of mathematics and engineerers. By not following the structured system, it will lead to disorganization and confusion around what to do. Something that is weird, since the agile methodology is the reaction to the bureaucracy of the overdefinisation on what to do and the lack of a focus on the end users.

The aesthetics of the traditional system of development is very clear. Just like with engineering and the workshops, its about the practial aesthetics of planning and order. It’s the emotional aestheic of saftey in a framework that defend you if timelines cannot be met and unforseen problems arive. It’s about creating a vison of a product and making it just like that. Which is a great thing, but there is a reason this way of development has slowly disapeard more and more. Which is saftey and comfortability does not create progress.

2.1.2 Agile methods

While the traditional framework might bring saftey in development, and a standarized way to make a product. The inventors and creators often work outside of that framework, and through the thought process that one should learn from ones mistakes. In this case, having the freedom to create something, find what could be done better and implementing those changes can lead to new and better ways of developing products.

A place we can see this type of approaches, are from the game development and computer clubs all the way back to the 1960s. This thesis will be focusing on the 80s, because it was around that time home computers became avaliable for everyone. The games and software created by these clubs did not follow the tradiontonal method of development, but was focused in creating something to entertain others or to send a message. One of these developers was Bernie De Koven, he and Jaron Lanier created Alien Garden in the 1982. which later has been termed the first art game. De Koven said in a interview with Jesper Jull, around the creation of the book “Handmade Pixles: Independent Video Games and the quest for independecy”.

> *“At this early stage in video games, cultural independence was slighly different from today: there was a sense of mainstream conventions, but De Koven also described working in open field of creativity freedom, especially on home computers”* (Jull 2019)

Which touches on the freedom to create something outside the established framework. This is not the only place one can find traces of what would become the adaptive agile method of development. The book “Gaming The Iron Curtain” by Jaroslav Švelch, takes a closer look at how computer and programming developed inside the iron curtain. His findings show how computers was introduced to Czech and the clubs and games that was born from this. The later part of the book, investigates how games and programs was used during the revolution to question and critique the society they were living in. Something they could do only through games because they were not seen as a medium capable of doing so. (Švelch 2018)

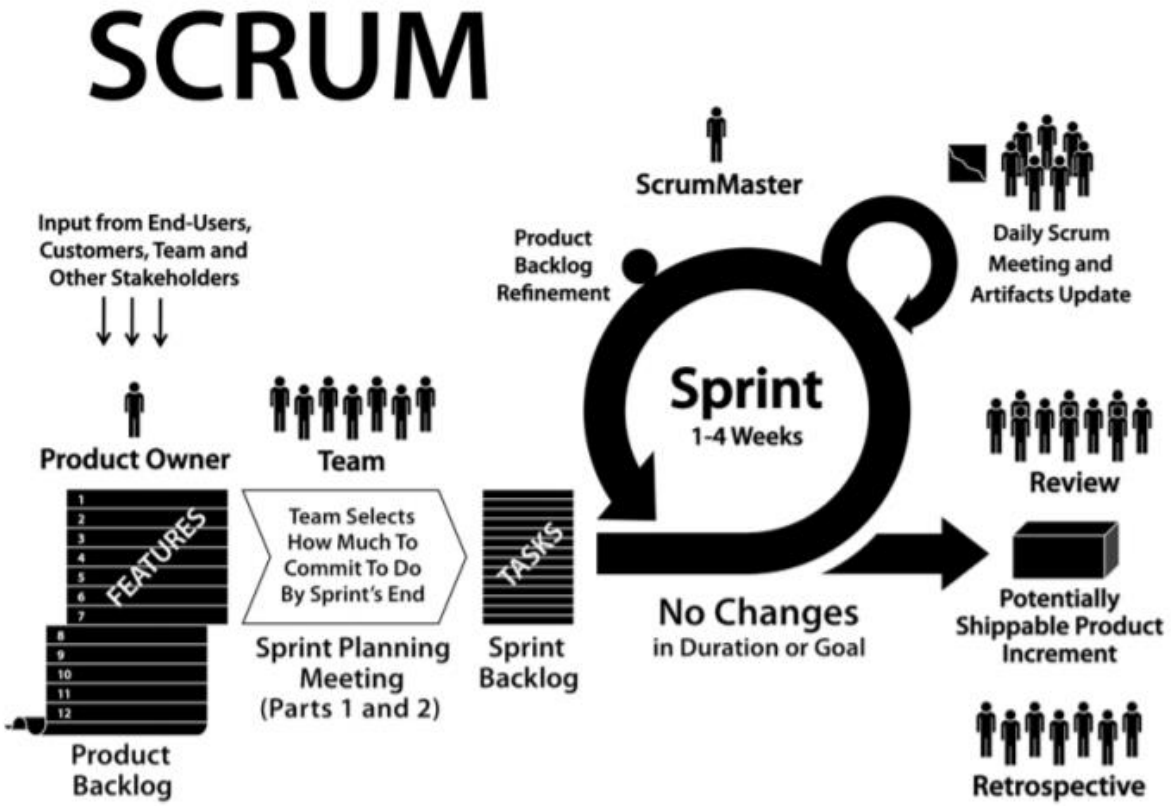
The point is that hobbyist, social critics and others often find themselves developing software and games outside of the traditional methodology. They focus more on the experience of the players or users of the software, while the traditional workshop view, is on just the designed product. The disconnect between the traditional methods and the creative side of the software field, creates discontent between developers and management, which led to the creation of the “Agile Manifesto” in 2001. (Grenning, et al. 2001)

This manifesto was created by the collaboration of seventeen people from different groups of development. These people had gone away from the traditional way of developing, to use a more adaptable and user focused methods, such as SCRUM and Extreme Programming. Their main goal was to create an alternative to the document driven and the heavy software focus that had controlled most of software development the last few decades. In February 2001, they met up over the weekend to discuss and create a frame for their views and thoughts on how development should change. This ended in the creation of the “Agile Manifesto” which is twelve principles that one should strive to reach for when developing. (Highsmith 2001) These principles describe the main values of the participants in this meeting and describes what they take pride in when working. They also created a sales pitch for the manifesto which is

> **“Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan”
>
> (Grenning, et al. 2001)

These few lines try to change the focus of what one should find important in development. It points out that instead of processes and tools you should see the individuals and the interactions happening. Let us not focus on documentation but rather on creating software. Let us try to help the customer to get the best product he can have and not negotiate with him and deliver something he might not need. Let us be adaptable and change when it is needed to, instead of planning everything and sticking to it. These ideals opened the doors for many developers to take back some control around how software should be made.

An example of a development method that has been gaining a lot of traction because of the agile manifesto is the for mentioned SCRUM. SCRUM is an adaptive and iterative approach on how to develop software. It is user focused and with the goal to deliver smaller increments of the product to test and make sure what is being made is what the user needs and wants.



(Sutherland and Schwaber 2011)

Looking at the picture above it might seem rather complicated in difference to the easy to read waterfall in chapter 2.1.1, but SCRUM is not that complicated. The way it works is that one creates a list of the features for the program, based on what the customer wants and what the users need. This is what one call the product backlog, or what is supposed to be made by the time the software is done.

This is followed up with a sprint planning meeting where the development team meets and discuss what tasks should be done by the end of the sprint, how much time each feature will take and who is going to be doing them. This creates the sprint

backlog. The sprint is just the name for the development process in SCRUM, it lasts for 1-4 weeks as shown on the figure. Every day or two the team sits down and talk about their progression and what hardships they have encountered. This is so everyone knows where each other work is at, and one might get the feedback that is needed to complete the challenges that appear. At the end of the sprint one checks if all the tasks are completed, talk about what they were able to do and not, the things that wasn't completed gets put at the front of the next sprint. This continues until the product backlog is empty.

When an increment is completed, it gets a "released" tag and is pushed out for testing by the customer and users. This is a major improvement from waterfall that could not show any progress before the software was complete. With SCRUM it takes 1-4 weeks to be able to show and get feedback. This happens in the review meeting at the end of the sprint, while the retrospective meeting talks about how they can improve the process for each team. This makes it so that the development should always be able to improve and adapt to different projects. (Sutherland and Schwaber 2011)

This type of development method also has a lot of communication with the end users. After every completed module or increment it will be tested with the end users to figure out if it is what they need. This makes it so that one might be showcasing a prototype every month, something that allows for both adaptability and collaboration.

2.1.3 Developers view on managment

Under the interview that was done in May/June 2020, the topic of what part of the development process they liked to work with came up. The product owners (PO) brought up communication as one of the main pillars of what they found interesting with their job. Developing for the user and trying to understand their needs and communicate these needs to the developer is a challenge they like doing. (Appendix A "PO 1", 3:00) As well as sitting down and focusing on the end user's problem and finding solutions to them. (Appendix A, "PO 2", 5:29) A follow up around this topic was what their thoughts around the agile development method was. The POs

thought that working this way was a much better because of the adaptability and control one has in the project. (Appendix A, “PO 1”, 5:15) If something likes Covid happens, they can change without the project taking too much backlash from the changes. One could also argue that with large projects, being able to develop and test smaller pieces and changing them if needed, is a great boon for both PO and the rest of development.

The project owners also brought up one of Warren Sacks claims. That society has another view on development of software. The conversation came up because of the question, have you experienced that the buyer disagrees with what is being made for the end user? One of the POs said that “We have been in some projects where the customer says they want something, but their lack of knowledge makes it hard to figure out what they actually are asking for”.(PO 2, 21:00) It was followed up by asking if agile would have helped with figuring out this knowledge barrier. The PO answered yes and no but brought up that the product focus of the customer can crash with the user focus that agile brings.

Customers often have a budget and would like to just get a price and hand off the requirements. Agile development does not work like that. With most development companies working with agile today, it is weird that their still acquiring work by winning bids on contracts. (Appendix A, “PO 1”, 16:05) To win a contract, user experience, unknown development time and price is not really a good sales pitch. The interview touched on this topic and it’s a clear example of the perception that still lingers around what software development is.

As mentioned, this is what Warren Sack was talking about. More specifically, by letting software development be seen as engineering and mathematics, customers and society, has gotten a perspective of what interacting with them should be like. Which sometimes lead to a culture shock when they meet. Based on the interview with POs, this culture clash is becoming less and less after having worked with software developers. By communicating and being clear on how software is being created, and why the focus is on the users, one can slowly change the customers perspective on software development.

The coders and designers that was interviewed, also had a positive view on agile development. By focusing on user experience, the designers have someone to focus

the software towards, as well as someone to test new concepts with. This is, based on what one of the designers said. What they find fun and interesting with this job was testing concepts and software with the end-users (UX 2, 34:50). One of the coders on the other hand, told a story about a project that had four hundred pages with requirements back in the day. They used three months to try to decode what the users wanted from that stack of papers. (Appendix A, “Programmer 1”, 05:35) By going agile this problem seems to have become mitigated. Instead of a stack of papers it is now meetings and discussion in person, something that from the programmer’s view is a welcomed change. With this change the programmer is now able to focus more on what they care about, which in this case, was creating technical solutions that helped the users.

The POs found the aesthetic in communication, figuring out solutions and making the people under them better. The programmers brought up the architecture and the technical solutions to the difficulties in the assignment. How should the software work, what feature and how are they supposed to work. The designers brought up the same things that the coders did for the most part. The difference is in viewpoint, instead of technical solutions and architecture they are focused on low fidelity prototypes (design) and finding out what the user will need and understand. They create solutions one can click through and test on the users. The thesis will look more in-depth at coding and design in the next two chapters.

The visual aesthetics in software management would be the conceptual models that only one of the interview objects said they did not use. These models describe the overall system or part of the system that developers are focusing on. In this way they are both practical and visual, a way one can think about models is as a visual representation of relations between functions. In the book “How to be a Geek” (Fuller 2017) they bring up the topic of object orientation (OO). The reason OO is being brought up, is that the models that is being talked about, often describes the world through objects. Every item in the world is an object that has relations to other objects it can interact with. This goes for systems, ideas, items and so on. One way to think of it is like sources. The sources in this thesis has a relation to the source list, which again is part of a book, who has relations to the authors and so on and on. The point is that modelling the world, or system in this case, describes how the system functions and how it should interact with its surroundings (other systems and

users). In a way, it is like painting but with lines and boxes. A good model could with other words, be visual pleasing to look at and give an understanding of the world and what is describes.

Communication, as mentioned, is also a part of system management. The question is what aesthetical properties does it have? Based on the interviews, communication is both emotional and practical. It is the medium that both emotions and practical thoughts are shared between those who are a part of the development process. Communication happens throughout the whole process and is one of the pillars of development. All the interviewees mentioned communication at one point throughout the interview and thought that it was an important part of development both for developers and towards buyers and users. With knowing that, making sure that there is an understanding of the thought process the coders, designer and user/buyers has, is important. Looking at traditional vs agile development one can see how the viewpoint of developers and management has changed the last decades. One can see from the interviews that has been done, that the developers views are closely linked with the agile way of thought. With it the aesthetics of the hobby and indie developer has appeared in business software development. If Warren Sack had released his book a few years earlier, it might have had a bigger impact than today. Most of what he is saying around engineers and mathematician's vs liberal art does not seem to exist, or if it does in a much more subtle way inside the development space.

The aesthetics of software management has also been rapidly changing together with the agile movement. Like mentioned, it has become more of a focus in looking at the users as people that needs help with fix a problem. The buyers is not buying a product, their buying the knowledge developers has around find solutions and implement it to help fix their problems. With this change in mentality they can be adaptable and creative without the constraint they had before. The interviewees did agree that there had to be some structure and planning, but it should not be the only thing they focused on. The focus should rather be on making something that will help the users, which is emotional aesthetic from the agile method

2.1.4 The Cultural Change

System management and project owners seems to have embraced the user focus that humanities and liberal arts often focus on. The problem is that companies and others have low to no knowledge about this change. Which is something that can create cultural conflicts when they meet to collaborate on projects. One good note is that there has been changes happening in school and healthcare that mirrors the agile movement. This is a has happened in a clear parallel and maybe even before the changes in software development.

In 1991, the Norwegian government came with a reform called “Ansvarsreformen” loosely translated as the responsibility reform. It was renamed HVPU (Helsevernet for psykisk utviklingshemede) in later times. This governmental change made it so that people with development disabilities was not going to be the responsibility of the county (fylke), but rather each district (kommune). This led to the institution that took care of people with disabilities to close. In its place. apartments were created that should make sure that the individual that was going to live there, got the tools they needed to live as normally as possible. Instead of the structure of an institution they should be given the closest form of normal life that is possible to give them. Where it is possible, they should be able to rule over themselves instead of following the plan made by the system (Kjøs 2020).

Ten years after the reform, a report from Ivar Brevik and Karin Høyland investigated how the service and living situation for people with development disabilities was after the HVPU reform. They found out that the living standards were high, and that service was more focused towards the individual. It was also clear that it was easier for people with disabilities to function in society, in the form of work and free time activities when having their own apartment. (Brevik and Høyland 2007)

The reason for pointing out this reform is to show the similarities between this and the agile movement. They are both the response to a system that does not look at the individual, that creates plans, structure and followed them to the smallest detail. These are systems that makes one detached from the users, in this case people with development disabilities and people using the software that is created. The solutions have much of the same thoughts behind them. The individual first, instead of

planning do something, collaboration with the users and change things to make it fit for the individual you are working for.

This reform laid the groundwork for a change in focus both in the health sector, but also in school. With HVPU saying that people with development disabilities should have an integrated and mostly normal life, schools became liable to give them the best education they could. In 1998 there was a change in the law for primary school, middle school and high school that changed who was available for special needs education. (Regjeringen, Om forholdet mellom «funksjonshemning» og «særlig behov for opplæring, 12.2 2020)

With this everybody that needs some extra help with their education, is by law, able to get the help they need. With this schools in Norway started focusing on developing individual education plans or IOP for those that needed it. In 2019 the government presented a new education plan. The new plan is by the words of the head of the department of education Jan Tore Sanner (translated from Norwegian) “Society is constantly changing, and schools has to change with it to make sure that children learns what they need to join it when their done (…) The new education plan will give the students a foundation for reflection, critique, to create and research” (Regjeringen, Nye læreplaner for bedre læring i fremtidens skole 2019)

What this mean in practice is that schools are now going away from a structure that said that these are the topics you must talk about. Instead its more up to the teacher to go as deep as they want into different areas. It is also a push towards doing more practical learning in the form of games, arts and crafts. Interdisciplinary courses are one of the things that are being looked at to make sure that students see the reason for why they are learning about a topic.

With other words there is a focus towards the individual, adaptability in the form of individual education plans, creativity, practical learning instead of theory and problem solving across multiple courses. As shown above these are thoughts that one finds in the agile movement as well as in the health sector. Something that the head of the department of education eludes to in his statement about the changes to the national education plan needing to follow the changes happening in society.

This thesis is therefore arguing that the gulf of incomprehension might have been identified by the Norwegian government and that his new education plan might be

what one needs to start making a change in how society views software development. As well as increasing the general understanding about how to create solutions that helps the most people possible.

.

2.2 Aesthetics in Coding

This chapter takes a closer look at the beauty one can find in coding. Both in the code and in the practise of writing said code. To do this the topics of structure, syntax, stylesheets and policies will be analysed towards what aesthetics in code are in a system development organisation. This chapter also brings up how aesthetics is being used as a tool to help programmers to understand and create better software.

As mentioned in the aesthetic definition, aesthetics is the judgment of beauty (Zangwill 2019). Following that logic, the aesthetics is different based on who is looking at it. This thesis is looking at how developers in a business looks at the beauty in code. This mean that programmers that work with digital art, digital poesy or other form of coding art will not be mentioned. Since the field of coding is so large, this thesis has chosen to showcase a few different topics within coding that shows the aesthetics in coding from the viewpoint of system developers in a business community.

2.2.1 Structure in Code

When talking about aesthetics in code, it is important to not just look at the code but also on the outside factors that pushes for code to be written a specific way. That is why the first thing this thesis will be looking closer at is the structure in code. More specifically the visual factors one can find in it, and what attributes developers find aesthetical, as well as practical factors.

When it comes to the visual part of code, the two terms that most often gets used is syntax and semantics. Syntax is the grammar of coding; it is the rules that govern how one are to write in a specific programming language. Semantics on the other hand is the meaning of what one is writing. Another way to look at semantics is that semantics is the symbolised version that describes how the software works. The reason for bringing this up is to make it clear that developers work around the syntax

to create the semantics. This means that when a developer looks at code, what they see first is the semantics of the code that is written.

An example of how different and still runnable code can be was shown in the article written by Nick Manafort and Michael Mateas “Obfuscation, Weird Languages, and Code Aesthetics”. Here they bring up the point that the syntax and semantics cannot stop the developer from making code hard to read. They post a hello world example to showcase this. This thesis made its own example using the principle they brought up. (Mateas and Nick 2005)

```
#include <iostream>
#include <string>

int main()
{
    for(int i = 0; i < 10; i++) {
        std::cout << "Hello world " << i << std::endl;
    }
}
```

This code posts Hello world 10 times once with a line break after each one. It is readable and understandable. In the Manafort example they started writing one letter each line, which is a semantic choice that makes the result harder to read but not the code. If one wants to make the code harder to read one could write it like this.

```
#include <iostream> #include <string>
int main(){for(int i=0;i<10;i++){std::cout<<"Hello world "<<i<<std::endl;}}
```

By looking at the two examples, there are a few rules of thumbs implemented in the first one that are not there in the second. These rules of thumbs are what coders often look after when they first watch new code. From an outsider perceptive, when code gets cramped together like in the last example, it becomes hard to understand where one-part ends, and another begins. To stop this, code is often written with indents to indicate different layers of the code. Think of it like paragraphs in writing. A text without paragraph gets harder to read because of the share amount of information presented.

Another part is to use whitespace to break up arguments to make it easier on the eye. These visual differences do nothing for the efficiency in the code but makes it much easier to test and read later. Most developers would agree with the two

examples that was shown here. There are though, many more rules that differ between programmers. That is why programmers and businesses has created their own style guides on how they want code to look.

Style guide or coding standards are documents describing how code, structure, comments and libraries should be used. It is the guide that every developer in a team should follow to make sure that the code their making looks the same. The reasoning for wanting people to follow style guides, is to be able to follow one set of standards when getting into the software. If every file in a project followed different standards, it would create tons of confusion for the developers that did not create the file. To look closer at the aesthetics found in these coding standards, this thesis is investigating the coding standard of Epics Unreal Engine (Epic Games 2020) and Googles C++ style guide (Google 2020).

At the start of Epics code standard, they give a few reasons for the creation of said standard. One is that most of the lifetime cost of a piece of software goes to maintaining it. Which means that it should be easy to maintain and easy to understand. Another part of it is that software is seldom maintained by the same programmer throughout its whole life. If it is not documented, commented and using a standard that is written down. It becomes increasingly hard for others to understand the software. If code is to be given out to others, it should be readable and easily understood. Lastly many of the conventions are required for cross-compiler compatibility. (Epic Games 2020)

Google on the other hand, define their style guide as “The goal of this guide is to manage this complexity by describing in detail the dos and don'ts of writing C++ code. These rules exist to keep the code base manageable while still allowing coders to use C++ language features productively.” (Google 2020) Google does mention a few goals as well for how their code should be. Focus your code towards the reader and not the writer, be consistent with existing code and the broader community, avoid hard to understand constructs and constructs that might act surprising. Lastly keep in mind the scale of the projects, when working on million-line software code, do not take shortcuts that might hurt the project in the future.

When one takes a closer look at what the rulesets are between the two, one finds there is a large difference in naming conventions. Unreal Engine used a letter in front of classes and variables to tell what they are. Example every Boolean should have a b in front of their names, for example bBoolean. This is something Google does not do. Instead they have different ways to write names based on the type they have. An example of this, is that constants needs a k before the name while other names follow the capital letter for every word in a name, kThesisName and ThesisName.

Even though there are differences the main thoughts behind these rules seems to be the same. By having a standard or style guide, everyone that approaches the code will have an innate understanding of how it should look and work. This gives experienced programmers a way to see if code is good or bad. They will base these assumptions on the community standards and the guide/standards that their place of work has implemented. These standards are prone to change over time as new solutions and the culture change.

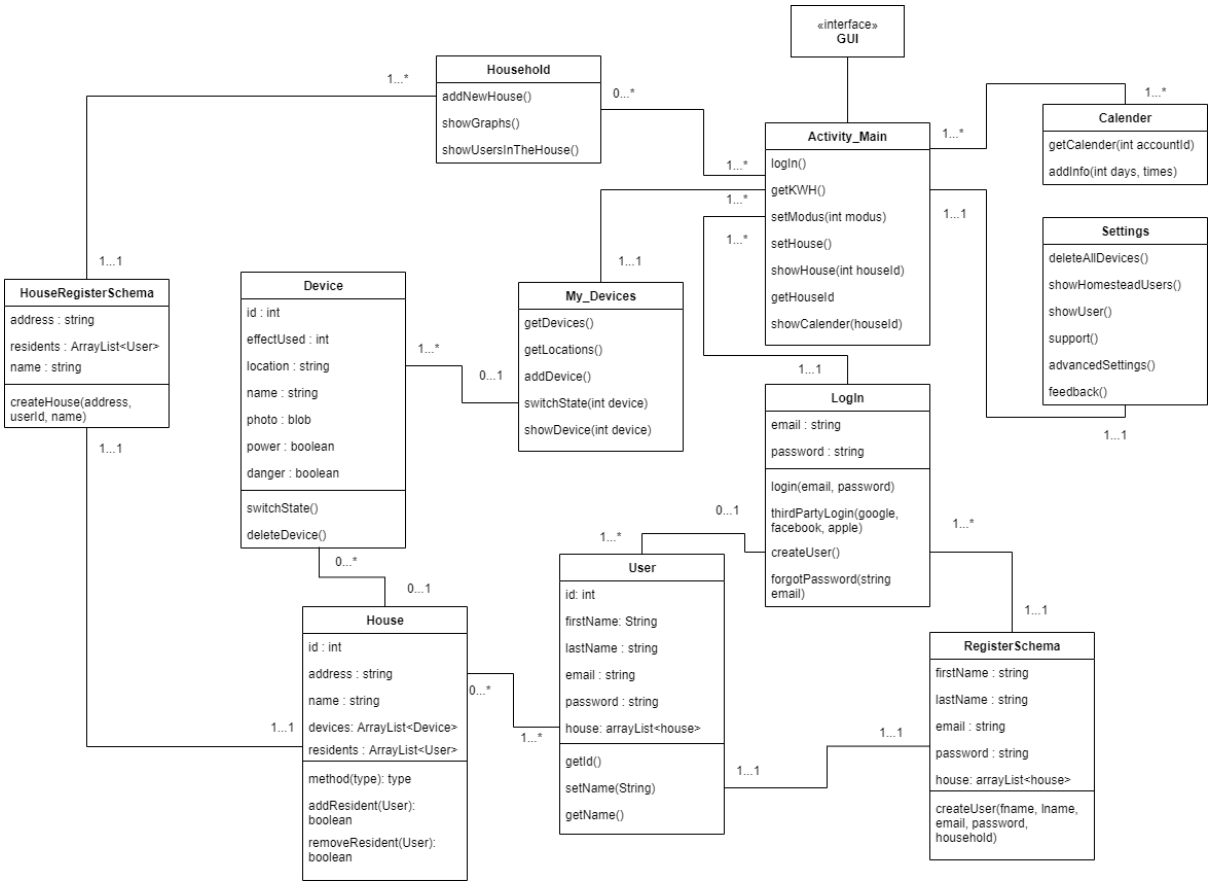
In 1999, Kent Beck and Martin Fowler released a shared chapter called “Bad smells in code” that focused on how and what one should keep in mind when creating code to stop it from “smelling bad”. By this they mean code that could be refactored or changed in a way to make it more readable, better documented, structural change and so on.

> *“One thing we won't try to do here is give you precise criteria for when a refactoring is overdue. In our experience no set of metrics rivals informed human intuition. What we will do is give you indications that there is trouble that can be solved by a refactoring. You will have to develop your own sense of how many instance variables are too many instance variables and how many lines of code in a method are too many lines.”* (Beck and Fowler 1999)

They go on to describe how to change your code to stop it from smelling bad. When reading through what they are pointing at one can see clear parallels to the coding standard and style guide of Epic and Google. “Bad smells in code” brings up duplication, large classes, shotgun surgery, feature envy and comments to name a few bad smells often found in code. (Beck and Fowler 1999) Which is something code standards and style guides help manage, by giving a set of rules to make coders aware of what they are doing. This thesis is therefore arguing that styles and

standards in themselves is a judgment of the beauty of coding. They are all created by communities that sees different ways to create code, and if one does not follow the guidelines the code will be smelly or ugly. As mentioned, they are there to make it easier to read and maintain the code which creates less frustration and increases the quality of the software. For the business software development viewpoint, the common attributes in the coding standard is the focus on readability, consistency on names and comments as well as remembering that others will be using the code one makes. One can therefor argue that all these topics are important for the aesthetics in coding and if any of these things are done half-heartedly it will leave a bad smell.

Another viewpoint that is important for the structure found in code is object orientation. As mentioned earlier, OO is a way to structure information into smaller more understandable packages and how objects relate to themselves and the rest of the world. (How to be a geek)



(Jostein Enes, Figure 3 “Object orientation” 2020)

The figure above is a class diagram following the OO thought process and is describing how a application that controls a smart house works. The way this figure

interacts with the coding structure is that every box is a class. The information found inside the boxes is the functions and attributes that exsist in the class, while the lines and numbers describe the releations between them. For example, Activity_Main, which is the home page, is connected to Household. The number 0…* means that Activity_Main has none to many household classes it can connect to, while Household must have one but can exsist for many Activity_Main instances (one household can have many users). Activity_Main on the other hand, does not need to have a Household but can be apart of many Households.

By making more classes that does specific features, one can create smaller files and classes that gives a better understanding of what they do. This means by proxy that it is easier to debug and change if needed. This works in tandem with stylesheets when it comes to creating readable and easy to understand code, as well as helping with moduling. It also helps that it is easier to fracture up the software so that one can create a feautre throughtout one sprint.

Stylesheets, semantics and OO is very usefull for programmers. These standards and methods helps to bring a readable structure, so that one can easly find faults in the program. For a programmer that is doing testing it is paramount that software code follows the same pattern and style throught the whole program. This makes it easier to notice inconcistencies, logical and semantic errors that might have happened.

When one hear “testing code” there are two things that one think about. Unit testing and reading code. Unit testing for those that do not know what it is, is a check list that gets programmed to make sure that the software returns the value the programmer wants to get back. This is done a bit different based on the language that is being used, but the thought is that if one sends in 3 and an 5 and the software adds them togheter it should return 8. If 8 is not returned the test fails. This can be done a lot throught the software, it helps both programmers and testers to find out different strenght and weaknesses with the code. The other testing methodolgy is reading code. Some busnisess do have quality assurance meetings where they read eachothers code. This is to make sure that it follows the standards they have and that there are no better solutions to whats been made. There are many other testing methods but therese are the most well known.

The reason for bringing up testing in the structure is that the stylesheet, semantics and structure is so important for testers so that it had to be specified. Chapter 2.2.3 will be talking more about how aesthetics and beauty interacts with the testing of code.

2.2.2 Work languages vs Machine languages

Warren Sack brings up two constructs he calls work language and machine language in his book “The Software Arts”. He defines work language as “To mean the language-the text- employed to describe the processes and products of work”. (Sack 2019) In a coding perceptive it means the code a programmer is writing, or the language developers use between themselves. For the point of this thesis it is the languages used for coding. Machine language on the other hand, is by Sacks defined as: “A machine language is a work language that employed in the design and analysis of machines. When a machine is designed to replace a human in a work process, the actions performed by the human must be translated into a machine language” (Sack 2019)

For developers the way they think about machine language is a bit different. Warren Sacks definition is missing a third step. A developer’s job is to turn human actions into their work language (code), which then creates the machine language that the machine uses. It is therefore the action performed by humans that is translated into work language, and then generating further into machine language. Michael Schmit described machine language as:

> *“Machine language is the language understood by a computer. It is very difficult to understand, but it is the only thing that the computer can work with. All programs and programming languages eventually generate or run programs in machine language”* (Schmit 1995)

A way to think about the difference between work language and machine language in coding, is that there is a translator sitting between the computer and the developer. Its job is to break down what the programmer says, “work language”, into machine code and translate the results the computer gets, back into a language or form that is

understandable for the programmer. Because the computer only understands machine code, one could look at the coding language and the compiler as the translator. When a programmer tries to create something, he might hit the limitations of what the translator understands. Think of it as translating paintings to words. This limitation has made it so that the work language of programmers does not look like human speech language at all. Even though coding languages has a big difference from normal speech, it has become more prominent that the they should be readable for most people that read them.

Like mentioned in the chapter 2.2.1. Readability has a lot to do with the semantic of how something is written, but it is not just that. Different languages have different syntaxes that can change how code reads immensely. As mentioned earlier, readability of code has a lot to do with the visual aesthetics of the code. Therefore, different programming languages might look completely different and focus on different aspect of what one could call beauty. An example of programming languages that tries to look like normal writing is COBOL. Instead of the math symbols they use words which makes it look much like pseudocode. This example comes from stack overflow

> *"ADD YEARS TO AGE.*
> *MULTIPLY PRICE BY QUANTITY GIVING COST.*
> *SUBTRACT DISCOUNT FROM COST GIVING FINAL-COST."*
> (Stack Overflow 2008)

There is other language that takes this even further like Inform 7, that writes as an adventure game, something that makes its very easy to create. As a drawback, the more these type of languages goes towards what one could call natural language (what we speak day to day), the more prone they are to misunderstandings and is often harder to debug if errors happen. On the other hand, the closer the work language is towards machine language, the more micromanagement of a computer's resources can be done. This leads to the code becoming more complicated to understand and write but giving more options to work with. The debate between languages, closely related to machine code and those that are close to speech, shows that there is a gulf between the science side and the culture side of software

development. With science focusing on the possibilities of the complex solutions, while the culture and creative side on readable and understandable solutions.

This is why software development most often use standardized and object-oriented languages like Java, JavaScript and the C languages. These languages find themselves in a middle road between machine and natural language. This allows them to not be too complicated to understand, while also giving the option to implement solutions at a level close to the machine language. This gives the programmers full control over the solution their crafting. As software developers the software that gets created will be different each time, and it is therefore a must that the language used allows for the creation of step by step solutions.

One of the programmers that was interviewed commented that

> *"There are many people with language education that do well in coding without math or engineering backgrounds[…] in the end, coding is about communication and writing, and if you are good at writing I think you can do it well within software development"* (Appendix A, "Programmer 1", 59:33)

This correlates with Warren Sacks claims that humanities or liberal arts is a main part of software development. The programmer does specify that it does not help with the deep technical solutions, but coding is much more than that.

Front end development requires less of a deep technical understanding and does work a lot like writing. One describes how components, for example a button, work and look in relations to everything else. In a way. it is a way of world building, of communicating how the world or software works.

The point is that languages like Java, C and others gives a toolbox to create what one could call worlds. So, if one was to talk about the aesthetics in code, one could argue that the unlimited potential of the medium is beautiful. For programmers, specifically, it is the fact that one can create features and code that removes inconvenience or fixes problems for themselves or others. This is reflected in the work languages that they use, most of them allows for detailed explanations of how one interacts with the world. Leaving them to describe it for the users of the software.

2.2.3 Beauty in Code

In 2012 there was a quantitative investigation around what developers found to be beautiful and ugly code. It was done by Brooklyn college and the graduate centre of the city university of New York. Their findings were put forth in the research article "The Aesthetics of Software Code: A Quantitative Exploration" (Kozbelt, Dolese and Seidel 2012). They sent out a survey to 12 experts and 38 novice programmers, to get a feel for the difference in aesthetics between new programmers and old once.

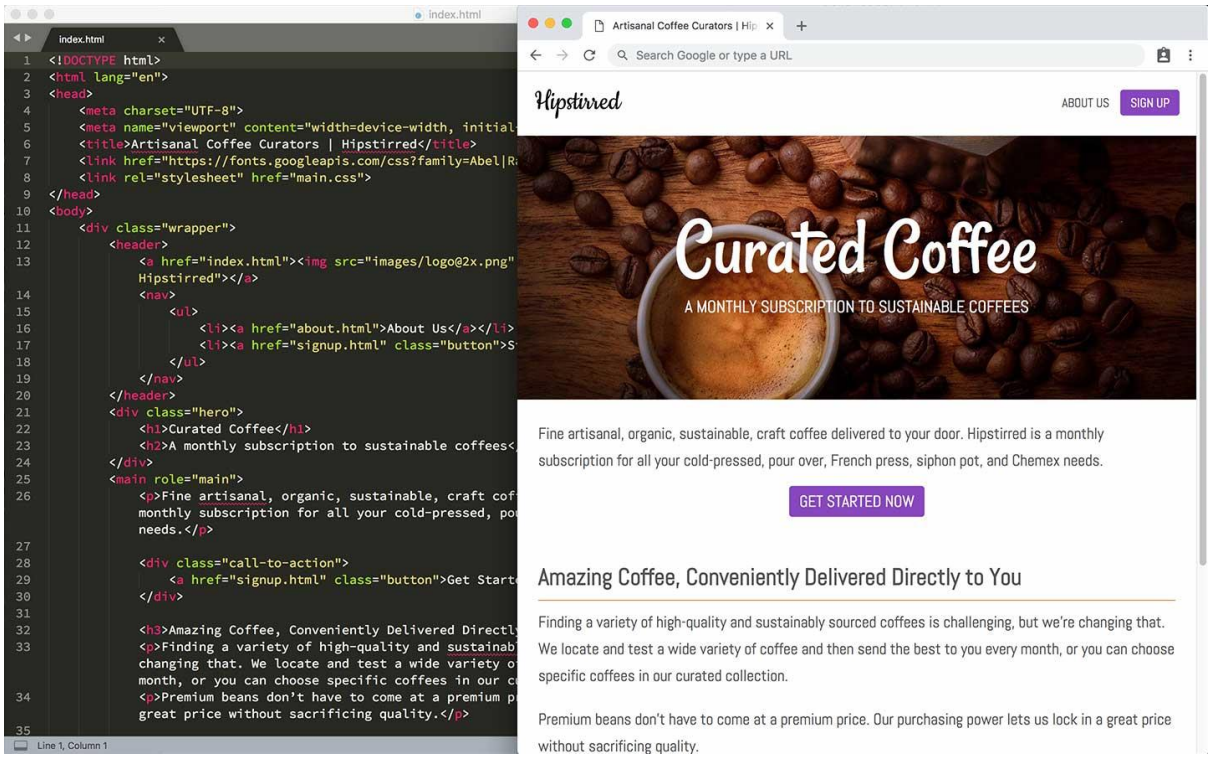
The results that was found from the survey was that every expert (programmers working for a business) had experienced an aesthetic experience when working with code and creative artefacts. They also found a correlation between functionality, beauty, ugliness, correct and incorrect code and aesthetics. From their findings, the time it takes to find and understand if the code is ugly is much quicker than with beautiful code. This fits into the other chapters of this thesis that has found out that practicality is one of the attributes programmers finds aesthetically beautiful. Based on this, one could see that aesthetics has a part in both debugging and testing of code. An interesting find is that ugly code seems to be a visual experience and is discovered by looking quickly at the code, while beautiful code needs more time and effort to be found. Correct and incorrect on the other hand, is closer to each other when talking about the time it takes to figure out if its correct or incorrect. With other words, the aesthetics of code is being used to make sure that the code that is being created holds a high standard.

With the stylesheet of how code should look in business, it is much easier to see if the code is "ugly". Which might explain some of the reasons why it is so much easier to find out if it is that then the other way around. Following up on this, the interviews done in this thesis also shows that programmers and other type of developers do tend to find problem solving to be one of the more fun part of the job. The programmers that was interviewed all talked about creating solution, and it was not only about the code, but the combination of code and other artefacts that made their jobs interesting.

Testing gains a lot from the aesthetic in code. As mentioned, it becomes significantly easier to test and read code that is not "ugly". As a by-product of this,

noticing bad or ugly code is a lot easier. The fact that it takes more time to figure out if code is beautiful, shows that what is beautiful about it might not be how it looks, but rather the artistic understanding by programmers that can see the nuances and the thought that has gone into a solution. Which is something that takes more time and energy to notice.

Another perspective is around describing the world and creating systems that work within it. Being able to create small eco systems of code in much the same way that figure 3 showed earlier is a fulfilling experience. Seeing the code that is being created form into a full system and tweak it into perfection, is something programmers often find interesting.



(Nobledesktop 2020)

Looking at the picture above, one can see the code on the left and what it creates on the right. In web development this is how one develops a webpage. The fact that one can see the changes happening as the changes happens, gives positive feedback to the programmer. As mentioned, being able to see the blood sweat and tears one puts into something take form and becoming something is a fulfilling feeling both emotionally and visually.

Looking back to the interviews, UX designers and front-end developers seemed to follow the same thought process. Their views follow the agile method of developing for the end user, and code is there for a part of the process to reach that goal. This thesis is therefore arguing the aesthetic feeling of working towards a goal, which in this case is the best software for the end user. This way of thinking is quite prominent in business software development. The view pushes developers to try their best when it comes to creating a sturdy and useful software that fits the users.

So, the aesthetics in coding vary a great deal based on who one questions about it. The main points they seem to agree on is that structure, language, readability and efficiency is important aspects to be able to say if code is beautiful. Everything else is up to the cultural background and education of whom you ask. Another way to investigate the beauty in code is through John Ruskin, who defined his view on art and the grotesque into three parts.

A) *Art arising from healthful but irrational play of the imagination in time of rest*
B) *Art arising from irregular and accidental contemplation of terrible things; or evil in general*
C) *Art arising from the confusion of the imagination by the presence of truths which cannot be wholly grasp* (Amigoni, Trodd and Barlow 2018)

The third definition has been redefined by Colin Trodd, Paul Barlow and David Amigoni as “the expression, in a moment, by series of symbols thrown together in bold and fearless connection, of truths which it would have taken a long time to express in any verbal way” (Amigoni, Trodd and Barlow 2018). This third form of art can be seen in science, mathematics and software development. For what is coding if not symbols thrown together in a bold and fearless connection of trust which would take too long time to tell in a different way? There is an aesthetic beauty found in there, of describing concepts and solutions with as few words or symbols as possible. An example of this is Einstein’s mass energy equation E= MC^2. The fact that it takes 5-6 symbols to describe how energy, mass and light relates to each other is fascinating.

This is the same in programming, the more knowledge and expertise one has around it, the easier it is to see the genius or beauty of code. Everyone that sees a painting can comment on what it looks like, but it takes knowledge to be able to see the brush

movements and understand the skills required to create it. By being able to put beauty and coding together, one can bring in the expertise from philosophers and researchers that has focused on this topic for centuries, to help create code that represents the world in a better way than it already does. For programmers in a business setting, it means that they would get new methods that might fit better when developing software for cultural purposes, it can also make it easier for creative people to be able to get into programming which would let them learn from each other and thereby grow, or help give a middle ground for global programmer teams to discuss and develop software with less culture clashes.

2.3 Software Design

This chapter investigates what aesthetics are for those that work with software design, more specifically user experience and user interaction. Software design is just a term this thesis is using to talk about these topics. It will also dive deeper into universal design and how this thought process has helped to change what designers find aesthetical. The chapter also explores how art and design, is being used to reach a standard, that allows the user to instinctively know how the software works.

2.3.1 User Experience

User experience is a term that encompass a lot of topics. Allistar Sutcliffe says in the book “Designing for user experience: Aesthetic and Attractive User Interfaces”:

> *“Hence in HCI, UX generally refers to a wider concept of design beyond functional products, which encompasses interaction, flow, and aesthetic design. It draws on literature from psychology, investigating how people assess aesthetically related design qualities, interaction and graphical design and contextual analysis of user experience.”* (Sutcliffe 2019)

With other words, user experience could be thought of as a greater topic around the design in software development, while user interaction is just that, how the user interacts with the software and the computer. Sutcliffe mentions that this definition is taken from HCI, which stands for Human Computer Interaction, and is therefore more focused towards the design and the structure of the software instead of code. There is an argument to be made that user experience should encompass the readability and structure of code. This is because following Sutcliff’s definition it encompasses interaction, flow and aesthetic design or if one looks at it from another perspective, readability, structure and semantics that is found in code.

As was brought forth in chapter 2.2. A lot of the standards and methods that are being used in coding, is to help developers with being able to interact, read and design code better. To make sure that other programmers can easily get into the

project by following the coding flow (stylesheets) developed by businesses. In the interviews with the different businesses in Bergen. The programmers that was asked had a big interest in user experience but did not look at it towards coding but rather as a design and methodology perceptive. When asked about how they tried to make sure their code was readable or if they did any tests around this, the answer was “I am the coordinator of one of the guilds in the business and every scrum team has their own code reviews and before every merge another team has to review it” (Appendix A, “Programmer 1”, 39:47). While talking about readability in code it came up that “We often have a smart solution and a readable solution to problems. And we try to use the readable solution where performance is not the key[...] with today’s languages and computers there are few places where the smart solutions are better than the readable once”. (Appendix A, “Programmer 1”, 43:33)

The programmer ended with saying that beautiful code is readable code and that code that is not readable is bad. The underlaying feeling that came out of this part of the interview is that user experience of code has much to do with the aesthetics of the code. It is therefore an argument that stylesheets, code standards and such, is an aesthetic artefact to help with the user experience programmers has with code. The topic of user experience in code, is not a topic that is often talked about with those words, but one can see a clear parallel between the goals UX and coding is trying to go towards. Looking back towards software management, most of what one can see in these fields, are following the thought of agile development and its focus on just these topics: Usability, customers first and less planning.

User experience is most often thought of as more of a planning and software development related topic, as mentioned in Sutcliffe’s definition. The UX Book by Rex Hartson and Pardha Pyla says that “Most in the field will agree that user experience, as the word imply, is the *totality of the effect or effects felt (experienced) internally by a user* as result of interaction with, and the usage context of, a system, device, or product” (Hartson and Pyla 2012). From a software development viewpoint, it is about the experience or feelings of those who use the product. It is therefore no perfect design answer when it comes to UX, since it goes on the individuals experience with the software. Jesse James Garrett follows up on this topic with

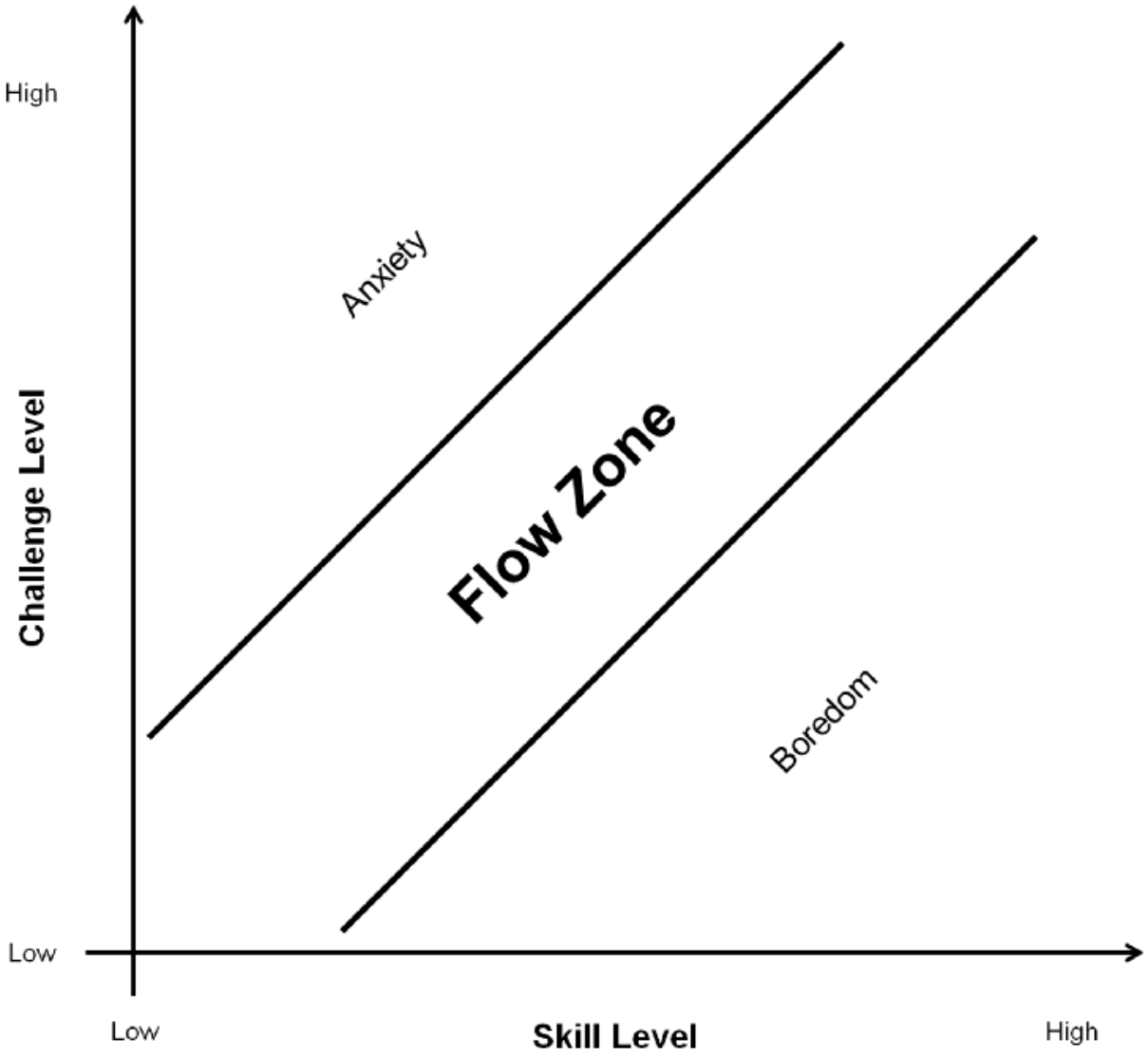
> *“User experience design makes sure that the aesthetic and functional aspects of the button work in the context of the rest of the product, asking questions like “Is the button too small for such an important function?””* (Garret 2010)

The goal is therefore, as Garrett insinuates, to find and create solutions that the user can easily understand and use. But to do that one needs to do what Garrett just did, ask oneself if what has been developed is the best it can be for the demographic that is going to use the software. This is the main thought process that UX developers and designers have when working. How would the user react with this feature? There has been done a lot of research around this. Everything from how people react to different colours to association of symbols. This has brought in many new people in software development that does not have a background from computer science or engineering. This has also brought in new perspectives and thoughts on how to develop for a better user experience. One of the programmers that was interviewed said “I do find it helpful with a project owner with a technical background, but it is just as helpful with one that has a greater insight into the needs and demographic we are developing the software for” (Appendix A, “Programmer 1”, 26:17). With other words software development is more than coding and technical solutions. It is about understanding and formulating solutions that is usable and workable for the end-users. So, the fact that people from with other educations and cultures are being let into the development process, will lead to the software being design for a bigger user base.

Because of the size of this topic, the rest of the chapter will be looking into some of the tools and aspects that software developers use to make sure the product holds the standard for user experience. From the aesthetic perspective of this thesis, user experience is one of the thing software developers find aesthetic. It is also the easiest topic to notice when it comes to beauty because of the focus on people, experience and design. When the UX and UI designers was interviewed and asked what they found fun, interesting or what passions they had. They answered user testing (Appendix A, “UX 2”, 34:50) and “creating a tool that people use, as well as gives them something that brings them value” (Appendix A, “UX 1”, 1:50). Both are topics that are closely related to each other. This is also the same thing the programmer said about problem solving.

User testing and creating something for the user, are both attributes of user experience. There is therefore room to say that the methods of user experience can be seen in an aesthetic light, since it gives an emotional response from working with it. If one was to pinpoint the aesthetic beauty in the top layer of user experience, it would be the beauty found in what it encompasses, flow, interaction and aesthetic design.

The flow of user experience is the thought that the user of the program must find a middle ground between interest and complexity. (Sutcliffe 2019) This is based on Mihaly Csikszentmihalyi, a psychologist that found out just that. When there are too much skill requirement people become anxious, when it is too easy, they become disinterested. One therefore needs to find the sweet spot where the users are interested and not to challenged too much. (Baron 2012) (Sutcliffe 2019) For software development this means that one could measure the time they take to do a task, listen to their experience with it and based on that slowly improve the software towards the “flow zone”.



(Orji, et al. 2019)

Interaction is the focus on the graphical user interface and how the user interacts with it. In “Adapting UX to the design of healthcare games and applications” one can find 8 criteria’s for efficient interaction design. These are based on L. Alben’s “Defining the criteria for efficient interaction design”. (Fanfarelli, McDaniel and Crossley 2018) These criteria are aesthetic experience, manageable, understanding of users, learnable usable, needed, mutable, efficient design process and appropriate. This will be looked closer at later in this chapter.

Aesthetic design is not user interaction, but rather the look and feel of the software. Sutcliffe says that:

> *“Design qualities such as good aesthetics and usability are likely to evoke positive emotions, such as pleasure and joy, leading to positive memories,*

> *although we tend to remember positive experiences in more general terms. We have found that a positive usability experience was not remembered in any detail but poor usability was, while general impressions of good aesthetic design were remembered favourably, so it appears that usability has to avoid serious errors; while investing in aesthetics adds value."* (Sutcliffe 2019)

A good example of this is when one finds an old website or new website that breaks with the conventions that the users already knows. The users then often get a negative experience with the webpage because of how different it is to what they normally use. It is the same whenever Twitter, Facebook or other large website changed the GUI, it creates backlash from people that says they do not like the change but forget about it a week later. One of the UX designers that was interviewed talked about intuition when it came to design. After asking what he meant with intuition he said following his gut feeling, what looks pretty or has some call to action for the user. He also mentioned it is very detail oriented, changing a colour could make the design work much better. (Appendix A, ”UX 1”, 21:10). On the other hand, the other UX designer talks about just that, how being the first out to change design towards a new feature standard might not be a good thing because people have not gotten used to it. (Appendix A, “UX 2**”**, 28:15)

When talking about experiences with websites, one is also much more likely to find less error in a website that looks modern, because if something looks outdated, we become on guard and starts looking for errors. It is therefore an important part of system development to create a beautiful aesthetic design to enforce positive experiences and mitigate the negatives ones.

User experience has a few ways that it is being used to make sure that what is being made gives a good experience for the user. One of these ways are scenario-based design (SBD). (Sutcliffe 2019) This is a popular method and has the same iterative approach as Scrum and most other agile methods. The way it works is that one gather information on the end, take their attributes such as ages and abilities, to create fake people called personas. After that a scenario is created on what problem these people have. A quick example is “Fred is about to head to work, but he has lost his car keys. This has happened before so he checks the web for a find your key application that might help him. The solutions he finds does not fit his needs and he must go back to manually find his keys” This gives the designer a way look into how

the users will be using the product and lets them create prototypes that builds on this. These personas and scenarios are the same as the user stories that developers are using for the features and coding but focused towards the software flow and design instead. What the designers are doing here is creating the framework for what the product is supposed to do and focus it towards the need of the users. The personas and scenarios is based on the knowledge designers have of the user, and often they go out to talk and interview the demographic that will be using the product to get an understanding of who they are.

The next part is creating storyboards, prototypes and testing. The way this is done is by a group of people that draw and create a paper version of the software based on the scenarios. They then take the best ideas of solutions and create a bit more professional version that they start testing on the users to get feedback on it. They then test multiple versions to find the best solutions to the different parts and after every iteration they evaluate it and try to improve it until the product is finished.

The aesthetical process of creating a product for a user is long and full of trial and errors. It is commendable how passionate the designers seem to be when it comes to creating the best experiences for the users. It is also important for the users and the buyer of the product to be able to be a part of the development process, and UX design focuses on keeping the contact frequent to make sure everyone has a say in what gets created. Therefore, the aesthetics of user experience can also be seen as the collaboration between everyone in the system development process, programmers, management, designers, users and buyers to find the best solutions and implement them in a way that is intuitive and aesthetically beautiful for everyone.

2.3.2 Universal design

As mentioned earlier, readability and usability are two attributes that often gets associated with aesthetics. It is therefore important to take a closer look at how designer develop software to be readable and usable for all users. Which brings us to the topic of universal design. As mentioned all the way back to chapter 2.1, the focus on agile development did not start in software development but in society itself.

The change brought with it an understanding that all members of society should be able to get entry and use public areas and products. It was these thoughts that started the focus on universal design. All humans no matter what abilities they have should be able to use normal day applications. It does not matter if one is blind, mental or physically disabled, young, old or anything in-between.

In 1997 the Centre for Universal Design at North California State University developed and released seven principles around what universal design is and what it tries to achieve.

1. *Equitable use.* The Design does not disadvantage or stigmatize any group of users
2. Flexibility in use. The design accommodates a wide range of individual preferences and abilities
3. *Simple, intuitive use*. Use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration
4. *Perceptible information.* The design communicates necessary information efficiently to the user, regardless of ambient conditions or the user's sensory abilities.
5. *Tolerance for error*. The design minimizes hazards and adverse consequences of accidental or unintended actions
6. *Low physical effort.* The design can be used efficiently and comfortably, with minimum of fatigue
7. *Size and space for approach and use.* Appropriate size and space is provided for approach, reach, manipulation, and use, regardless of the user's body size, posture, or mobility (Null 2013)

These seven principles are still being used today and is still very relevant. An interesting thing is that universal design became a thing around the same time that software development started thinking about agile development. There are also clear parallels between the two, like their focus towards making sure that what is being developed is for the best of the end users. These design changes are for everyone, a great example for this is the change from doorknobs to levers. By changing to levers it makes it so one can open doors without using the whole hand. So, if one is

coming from the store with full hands it is possible to get the door open if it is a lever but not if it's a knob. (Null 2013)

For software development, universal design has become a part of the design process. It helps with creating a robust design that leads to less errors and increases the user experience for everyone. In 2013, Norway passed a law that said that all information and technology solutions had to strive towards universal design, so that public solutions could be used by everyone. (Regjeringen, Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger 2013) This led to software designers in Norway having to stretch themselves to upgrade and change a lot of applications to reach the standards of this law. Last year (2019), the European Union passed a directive to make sure that the accessibility requirements around products and services is fulfilled by every member. (Parliament 2019)

Before these laws was implemented organisations, governments and others had started a collaboration to create a standard for web content accessibility called WCAG. WCAG was published in 1999 and was updated in 2008 to 2.0 and in 2018 to 2.1. (Lawton, Web Content Accessibility Guidelines (WCAG) Overview 2020) Something that fits the timeline of the agile movement and the general change in society that happened around that time. Either way. WCAG is a standard that defines

> *"How to make Web content more accessible to people with disabilities. Accessibility involves a wide range of disabilities […] These guidelines also make Web content more usable by older individuals with changing abilities due to aging and often improve usability in general".* (Lawton, Web Content Accessibility Guidelines (WCAG) Overview 2018)

These are the same thoughts and principles that defines universal design, the difference is that WCAG also has success criteria's that must be tested to confirm that it is up to an acceptable standard. The results from a WCAG test is A, AA and AAA, where A is lowest, and AAA is the highest. One of these tests are based on colour and the readability of text. Webaim is a web page that lets one choose the colour of the background, and the text and creates a contrast ratio and checks it up towards the WCAG guidelines for colour and text. (WebAim Contrast Check 2020).

By playing around with the colours on that webpage, it gives an understanding of how text and colour has a huge impact on how easy it is to read information on the web. It leaves one to think about how small changes might have a greater impact on user experience then one first thought.

The WCAG is an interesting standard/guideline that helps designers to create what one could describe as aesthetical good choices for software design. Much like the stylesheet and standards mentioned in coding, it helps and gives a framework that should be followed to make sure that the most users are able to use and gets a good experience with the product. The greatest difference is the fact that this is a global standard instead of a bossiness standard and that the WCAG does affect how one should code but not the other way around.

Universal design is by itself an attribute of aesthetic in user experience. Sitting down and micromanaging design to be able to make it as helpful for as many people as possible brings with it its own kind of beauty. One of the UX designers said that “universal design makes me simplify the GUI and make it more general. So even if I do not think about the outliers it helps with forming the product for everyone.” (Appendix A, “UX 1”, 39:50). This shows that universal design is a framework to help form and create a good design, as well as an aesthetic perceptive focused on making sure that everyone can enjoy the product.

2.3.3 User Interaction Design

Then there is user interaction. User interaction is, as briefly mentioned earlier, is about how the user and the graphic user interface interacts with one another and how to make sure that this leads to the best experience possible. As written in Interaction Design: Beyond Human-Computer Interaction: “It is important to point out that one cannot design a user experience, only design for a user experience. In particular, one cannot design a sensual experience, but only the design features that evoke it” (Preece, Rogers and Sharp 2002). User interaction is trying to create features and graphics that gives their users specific feelings when they interact with

it. They are also focused on creating a good experience for all users and is therefore using both user experience and universal design to reach this goal.

Dan Saffer comments in his book “Design for interaction: creating innovative applicatios and devicces (voices that matter)” that:

> “Interaction design is about behaviour, and behaviour is much harder to observe and understand then appearance. It is much easier to notice and discuss a garish colour than subtle transaction that may, over time, drive you crazy” (Saffer, Designing for Interaction: Creating Innovative Applications and Devices (Voices That Matter) 2009)

One can from this understand that user interaction and indirectly the WCAG is trying to figure out human behaviour and their reactions to colours, placement and thought process, when it comes to software. It is the designer’s job to focus on what the users want and need, even though the users might not know it themselves. This leaves them with a job to create and find solutions that has no end answer, there is always something to make better or try out. Saffer says that designing is not about choosing among different options, it is about finding and creating a third option that is better than the others. (Saffer, Designing for Interaction: Creating Innovative Applications and Devices (Voices That Matter) 2009) Which is something that gets touched on in the interview with one of the UX designers. (Appendix A, “UX 1”, 18:13, “Solutions design”) They talk about having to change the design that was working p10erfectly because of outside factors. This is not a good feeling but something that happens, but they still find it one of the more interesting parts of the job.

The reason why user interaction is being mentioned by itself is that it is one of the topics that gains the most from a liberal art perspective and aesthetics. The user centric development and the use of sound, colour, systems and more, to create emotions and connection between the product and the user is some of the things that makes user interaction and user experience interesting from this thesis viewpoint. As mentioned earlier, Saffer says that design is about behaviour and how to create a framework that allows connections and feelings to be generated in the users. One could look at it as closely related to actor network theory (ANT) (Latour 1996), about how the designer, structure, graphical user interface and the user is

connected and changing based on each other. More specifically for this thesis it would be how the software designers are able to create relations between experiences, colours and other parts to connect to similar experiences in the user.

This is also an ever-changing topic because of the advancement in technology. One of the UX designers said “What is kind of like a reality check after a few years is that everything changes. Technology changes, people change, how to use a phone change. What we wish is to follow the trends as the come and go. To find and follow a balance between trends and innovation” (Appendix A, “UX 2”, 28:15) Showing that this is something that is an important topic to keep in mind.

For a software development business, it is important, as mentioned by the UX designer, that the user interaction design can follow all these trends but still know when and what to innovate. The balance that was mentioned is the flow zone from the user experience part of the chapter. There are clear aesthetical points in user interaction, from the practical aesthetics of ANT, that shows the human relations between developer, machine and users. And the emotional aesthetics of putting oneself in the shoes of the users to experience, what they feel so that one can slowly change these feelings by editing the design.

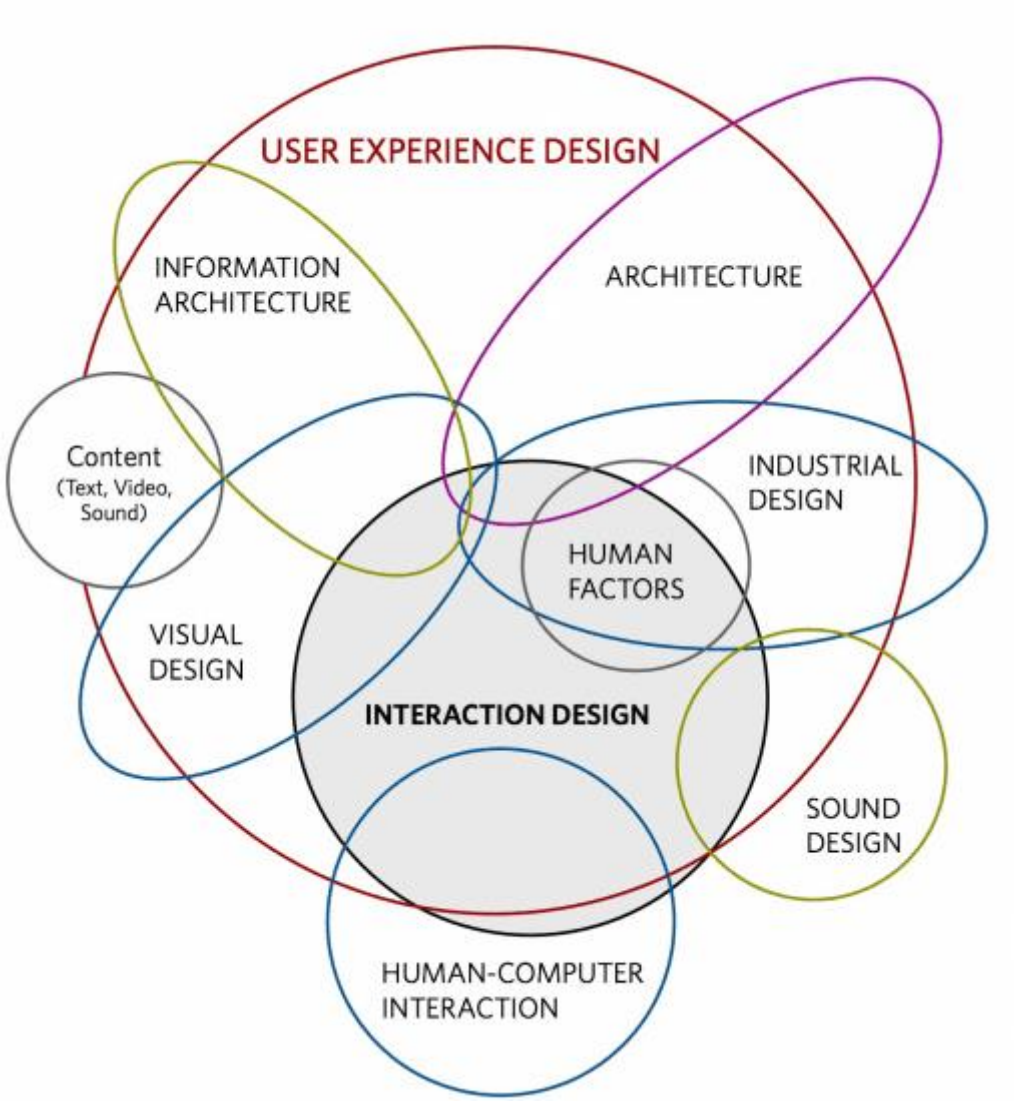
When developing the GUI, one has to mention the fact that they are working with visual aesthetics. Symbols, boxes and pictures are important aspects of this. There are also been done a lot of research about these things in other fields of study that can and should be useful. Art and linguistics have had hundred even thousands of years of time to test and develop their aesthetics and user interaction design can learn much from these fields of study. As mentioned earlier, one of the programmers said that other fields of study give a different understanding around the topics that their developing, and that this knowledge helps with developing the product just as much as another computer scientist or engineer.

To get a better understanding around this, one can look towards interaction design in games, more specifically Candy Crush and other microtransaction apps. The way they are designed is with the strategy that the players should get enough good feedback in the start, so they get invested in the game, before they turn up the difficulty. Following the flow theory mentioned earlier. User interaction design is what keeps the interest, this is in the form of colour pallets that trigger good feelings,

explosions with “Good Work” and similar slogans. This is all examples of how aesthetics affects the end users and can make them, in this case, use money by making them used to the feeling of mastery and then slowly start taking it away while offering insignificant sums to make them regain what they lost. (Hart 2017) For software development in businesses this is not applicable, but it gives an understanding of how important and how powerful user interaction design can be. The fact that the design that is being created has so much effect on the users is both interesting and terrifying.

2.3.4 Aesthetics of software design

After having taken a closer look at user experience, universal design and user interaction design one can see how they interact with each other more clearly. Saffer tries to show how interconnected UX and UI is by creating a figure to show what they focus on.



(Saffer, Figure 1.15 The disciplines surrounding interaction design 2009)

As one can see on the figure UX design is a much broader topic. It both design, coding and architecture of the software, while user interaction design only focuses on the design that the end users are going to interact with. This gives us a better understanding of how project owners, coders and designers must work together to make sure that the product becomes user friendly. This is not a job just for user experience designers and user interaction designers, but everyone in software development. It is a team effort to make sure that all the clogs in the system has been thought of and designed in a way that not only makes the software intuitive for the end users, but also for those who are going to develop on it later and maintain it. From the interviews that was done with UX and programmers, it came forth that they work in teams and help each other with designing the product. (Appendix A, “UX 1”, 23:55) On the other side the programmers showed a large interest in universal

design (Appendix A, “Programmer 1”, 54:45) and user experience. Based on this one can see that the gap between engineers and designers seems to not exist in the businesses this thesis looked at. It shows on the other hand an understanding on both side that working together is needed to be able to create something beautiful, and that could be argued to be aesthetically beautiful.

Following up on this, one of the parts that was brought up by the UX people was that it was much easier to work when sitting together with their team. The team they were talking about was the scrum team, which has both project owner, programmers and designers. (Appendix A, “UX 1”, 23:55) By being able to freely communicate with each other, the UX designer was able to focus on creating something then quickly get feedback from the other developers. Being able to see how the other parts of the development works allows for new perceptive and inputs. The way the interviewees talked about this gives of a feeling that it takes the whole team to push and develop with the user in mind. Something that is reinforced in the literature and latest by the figure from Saffer. It is this collaboration and communication that the project owners mentioned as the most interesting part of their work. With other words team communication, relations and teamwork is one of the more aesthetically pleasing things with how software development is being done today. And they go hand in hand with user experience, because for software development the users are the project owners, programmers and designers.

There a difference in culture between games and business software development as mentioned in 2.1. This different can be seen clearly in the end products. There are many games that use microtransactions while business development does not have this. One could look at mobile games and their focus on microtransaction and the design that give of the feeling of gambling, something that more and more countries is starting to agree is unacceptable. Latest country to try to do something about this, from when this thesis was written, was Canada suing Electronic Arts over loot boxes. (Kent 2020) Based on how prominent these types of mechanics are one can imagine how lucrative and damaging these are for people. From a user experience perspective, it is not in the interest of the users to develop a product like this. For not to mention the ethical shadiness of it all. As the digital continues to become more and more important, this thesis argues that the focus on what is best for the end-users becomes more important as well. As shown programmers, developers and

project owners do share the view that a user focused goal is something they are passionate about. It is also important to cultivate these feelings to create a culture that does just this, so that what is happening in the games industry does not affect all software development.

There has been a lot of talk about the user experience of the end-users and programmers. And new tools are being created for designers, more specifically sketches to code generators. This will allow designers to learn minimal code and focus more specifically on the interaction with the graphics of the software. From the aesthetic view, it will be interesting to see how interaction design will change if these solutions become good enough to generate design based on the standards of WCAG.

There are already websites out there that are pushing tools that should in theory allow non software developers to create their own websites by drag and drop. (Squarespace 2020) The thing they are missing though is the expertise that a software designer would bring with them. From a practical aesthetic point of view, interaction design that is so simple that one is able to drag and drop seems to be the goal. Minimalistic, easy to use and understandable tools and design, is part of universal design. In a way, software design has gone full circle and started to implement universal design in software development, which is something that was needed. It is important though to keep looking at software development with a user experience mindset to find inconveniences, like the sketch to code process, that one can fix. For that is the beauty of software design, creating solutions to remove these annoyances.

2.4 Conclusion

The conclusion of the thesis will go over the findings and try to summarize what the aesthetics in software development is. It will also look at the research question and the findings of the thesis, look at the use of aesthetics and argue for why more research on the topic is needed and what research and work should be focused on further.

2.4.1 Aesthetics in software development

Throughout this thesis, the focus has been on the aesthetics in the topics found in software development. This time it will be looking at it from a bigger picture and try to connect everything together to give a better understanding of relations between software management, programming, design and the aesthetic similarities.

The goal of software development as defined in chapter 1.1, is to create solutions to problems or inconveniences with the help of software. Creating and solving problems is what software development is about, and how this is done is by focusing on the details, communication and working as a team. Something that project manager, programmers and designers all have mentioned as thing they find important and interesting to work with. This thesis argues that the aesthetical judgment that the developers have around creating and solving problems is a practical and emotional one. By focusing on problems and creating ways to mitigate or solve said problems, software developers tend to figure out how the situation works and then create a feature to make it “easier” to handle. So, the creation of the solution can be a practical aesthetic while the underlaying reasons for wanting to solve it is emotional. When talking with the interviewees they used emotions to describe their thoughts and reasons to like or dislike different aspects of development. Something that is one of the forces that drives them to specialise in different technical parts of the process, some likes communication, some technical solutions, others problem solving. With all these different views, it is important to be able to see what others find aesthetical

about their specific specialisation. This is so that one is able to work together and make the job easier for everyone.

And that brings us to another topic that has been mentioned in every chapters so far, user experience and user focus. As talked about in 2.1 and 2.3.2, there has been a social change in how society and software development views the users of software, and more generally products and solutions. The things that was acceptable before the millennium, is no longer how thing should be solved. The whole thought process that was steering society, started to focus more on how to help and develop products and solutions to help everyone and not just the able bodied.

As mentioned earlier, this social change started in Norway with the healthcare reform and the focus towards universal design. This, together with the growing support for agile development methodologies, was able to change most businesses towards an agile way of thinking. The reasons for bringing this up is that the user centric view of developing for the users is strongly embedded in software development. The focus on developing a quality product that is there to help the user is a noble thought that all the interviewees mentioned was something they were striving towards. The fact that developers focus on what is best for the end-users, gives a perceptive into the ethical and philosophical viewpoints they have. This thesis argues that the ethical, philosophical and emotional thoughts that developers has around software development is its aesthetical judgment. It is these judgments that shows what topics, solutions, processes and ideas is seen as beautiful or not.

Morkel Theunissen took a closer look at what professionalism is in software engineering. He points to the five attributes IEEE mentions: One, a professional software education, two, a voluntary certification or mandatory licensing, three, specialized skill development, four, continuing professional education, five, communal support via a professional society and five, a commitment to norms of conduct often prescribed in a code of ethics. (Theunissen 2009) At the end he specifies his view on it.

> *"One may naively state that the first four points are primarily an infrastructure and process problem with an associated solution. However, for practical purposes one may state that the crux of professionalism for the individual lays within the last point —"A commitment to norms of conduct often prescribed in a*

code of ethics". This brings us back to the basics of humanity, the values that are embedded in the fibre of the individual which might be shared to some degree by a group/community." (Theunissen 2009)

As Theunissen describes professionalism, one can see it has a clear parallel with the aesthetical beauty found in software development and in the different topics that has been brought up. Based on this one can see that aesthetic beauty and professionalism are closely related. Since professionalism is a commitment to norms of conduct, one can see how following the norms might be aesthetically beautiful while breaking with them creates a negative aesthetical response. Something that is shown with the norms and standards mentioned in this thesis and how developers think and feel about them as well as what beautiful code is and is not.

2.4.2 Research question

Throughout this thesis the focus has been on the aesthetics of software development, in both small and bigger topics. Looking back at the research question **"How has software development changed since it was introduced and in what way has aesthetics been a part of that change",** one can see a clear change both in development and the culture surrounding software development. As Meisenberg brought up, there has been a gulf of incomprehension between mathematicians, engineers and the other liberal arts. (Meisenberg 2018) Based on the findings of this thesis that gulf seems to be much smaller today.

With the focus going away from a waterfall method towards the agile methods, it brought with it a humanity perceptive that made software development stop thinking about developing just the requirements, but rather making sure that it was useful and what the users of the product needed. This change with the societal changes towards universal design, made it so that the value of user experience (UX) and coding had to be re-evaluated. By creating Scrum teams that had people from all sides of software development and increasing the need for good design it slowly started chipping away at the notion of the gulf.

As pointed out in the chapter 2.1, this change was influenced by programmers and designers outside of the mathematical and engineering circle. Those that created programs for fun, by themselves or in small businesses. The reason for why smaller business and single developers have another view is because that they had to design and develop everything by themselves. Something that gave them a whole different view on development. There is beauty in both design and programming, something that is easy to miss when focusing to hard on one's own field and not the bigger picture. This is what the agile movement brought to software development, the perspective to see the whole picture. An ethos to follow, that gives the developers enough freedom to do what they feel is right. To push the field further by experimenting and working together in a way where everyone is working for the same goal, the best product for the end user.

When it comes to aesthetics it has been a large part of this development. Looking at the change that has happened, aesthetics is one of the reasons developers felt that waterfall methodologies was wrong. It took away their ability to be creative and change the software as needed. It also had a focus on documentation to such a degree that developers were burnt out before beginning. An example of this is the programmer that was interviewed that talked about the 300 pages of documentation that they needed almost 6 months to get up to speed on. This does not feel like the right way to do things. So, when the software developers sat down and decided these are the aspects that we find important, it spread like wildfire. This shows that there was a bad aesthetic feeling in the field of software development, and by changing the methods and thoughts on how to develop it brought a new passion to it.

This thesis argues therefore that aesthetics in software development is one of the leading actors in changing and creating standards for the field. From software management to coding standards to design, the thing that is the same is the aesthetic judgment of the developers and how it reinforces the work that is being done. As the thesis has shown through its interviews and literature search, aesthetics is interlinked with all aspects of development. One can therefore argue that a cultural change around what people find important, will influence software development just like in the early 2000s with the agile movement.

2.4.3 What does aesthetics bring to software development

What use has software development for aesthetics? As shown and talked about, the aesthetics has always been a part of the creative process. From describing the world to physically creating an object. What makes it so important in software development specifically, is that it is both of those things. Aesthetic judgment helps with telling the developer if what they are seeing is up to standard or is breaking with the for mentioned ethos of software development. This is the reason why ugly code or design is easier to notice then beautiful one. Aesthetics does therefore bring a way to see what code, design and architecture, is by the majority, seen as beautiful, of quality or efficient and maybe more importantly what is not.

Another thing is the professionality and how one appears towards buyers and end-users. As talked about earlier, the buyer of the software might not know how software is being developed and is just looking at the price and their own budget. Since software is a growing product it might therefore lead to confusion and problems between software developers and the customer. It is therefore important that it gets handled in a professional manner where software developers tell them how it works. Aesthetics can be a part of this communication. Showing and telling them how and what is being delivered and why it is being done this way can lead to the collaboration that developers are trying to reach for. It might also slowly change the view these different businesses that buy software, has on how software is developed.

It might also be smart to try changing what students are being told about software development. This is to allow more groups and not just data scientist and software engineers to take the path towards software development. A good way to do this is by showing them the aesthetics of software development and how creative studies can be used to enhance the development of the product. That software development is much more then coding, as they are told today, and that other liberal art fields are also able join these businesses and do well. This thesis also argues with its findings that creating Scrum teams across different fields of study to allow students to see the practise of their education can help them understand their positions better. That what

their being thought is not to make programs but how to solve problems with the help of programs and other tools in their possession.

2.4.4 Future Work

As mentioned at the start, this thesis began because Warren Sack did not look at the whole of system development but rather just programming in the book “The Software Arts”. In a way, this thesis is a continuation of his work and a springboard for others to investigate the connection that’s been found. This thesis could have focused on a single topic and gone deep into that. Instead this is the groundwork to understand how interconnected aesthetics and software development is and give others the tools and inspiration to look closer at aesthetics.

The first part that could be continued is looking deeper at the aesthetics of coding and talking with more programmers, both in businesses and outside of it, on what they find aesthetical about coding. There is a cultural difference between businesses, game developers, hobby developers and others. It would therefore be an interesting assignment to investigate the differences found in these groups and how it affects the product that gets made. Based on the findings of this thesis one can already see that there is a difference in the management of software in large scale game companies versus software businesses. A further study on this could give a better understanding of why this is and give a better understanding of the different programming cultures out there. It could also go much deeper into the differences in stylesheets and investigate how the programmers feel about the rulesets they must follow.

Another topic that could be looked further into is testing. Testing was mentioned to be one of the areas that gained a lot from aesthetics, but it was never talked much about. A future study around testing and checking how and if aesthetics is being used for this purpose could be interesting. It could be a way to support the findings in the programming chapter, that aesthetics able to decrease the number of errors and increase the readability of the code.

Since testing also can be done on the UI side it could also be looked at towards that. How is testing being done in UI and is the aesthetic judgment of the designers different from the users? This would let the study figure out what aesthetics the designers have and how it conflicts with the people they are developing for. This could be a good digital culture thesis on just that. How the aesthetics of developers affect the users. A way to do this is to interview the users on what they want, what they like and such before the designers show them the product and see how their views has changed afterwards. It would draw on McLuhan and his thought that “The media is the message” to showcase how our perception is coloured by the technology we use.

While talking about the how users and developers affect each through technology. This thesis brought rather early up Sacks claim that school and education still is under the thumb of the science department. And as the thesis argues, one of the ways to fix this is to teach the customers and students about what development really is. From this thesis one could start a study about just this, finding out what customers of software development think it is, as well as from the viewpoint of high schoolers that might be thinking about joining a software development program.

On the other hand, one could also go deeper into what this thesis mentioned as professionalism. What is a professional programmer and what does it entail? This would be a study around what qualities developers find important and what image they are trying to reach for. It could then look at how the image is seen by customers, end-users and students. One could also look at the different meanings’ professionalism has between groups inside of development or even between different programming practises. Since this thesis argues that professionalism is manifested by the aesthetics judgment of a group with the same views, it would be interesting to see how different this view can be between hackers and programmers.

Sources

Amigoni, David, Colin Trodd, og Paul Barlow. *Routledge Revivals: Victorian Culture and the Idea of the Grotesque (1999).* Routledge, 2018.

Baron, Sean. *Cognitive Flow: The Psychology of Great Game Design.* 2012. https://www.gamasutra.com/view/feature/166972/cognitive_flow_the_psychology_of_.php (funnet 10 10, 2020).

Beck, Kent, og Martin Fowler. *Bad Smells in Code.* 1999. http://www-public.imtbs-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/BeckFowler99.pdf (funnet 09 15, 2020).

Brevik, Ivar, og Karin Høyland. «Utviklingshemmedes bo- og tjenestesituasjon 10 år etter HVPU-reformen.» *veiviseren.* 2007. https://www.veiviseren.no/forstaa-helheten/forskning-og-utredninger/rapport/utviklingshemmedes-bo--og-tjenestesituasjon-10-ar-etter-hvpu-reformen (funnet 09 25, 2020).

Carroll, Noël. «JSTOR.» *Journal of aesthetic education 19, no. 4*, winter 1985: 5-20.

Epic Games. «Coding Standard.» *Unreal Engine.* 2020. https://docs.unrealengine.com/en-US/Programming/Development/CodingStandard/index.html (funnet 10 05, 2020).

Fanfarelli, Joey R, Rudy McDaniel, og Carrie Crossley. *Adapting UX to the design of healthcare games and applications.* 2018. https://www.sciencedirect.com/science/article/abs/pii/S1875952118300211 (funnet 09 20, 2020).

Fuller, Matthew. *How To Be a Geek: Essays on the Culture of Software.* Polity; 1st edition, 2017.

Garret, Jesse James. *The Elements of User Experience: User-Centered Design for the Web and Beyond.* New Riders; 2nd edition, 2010.

Google. «Google C++ Style Guide.» *Google github.* 2020. https://google.github.io/styleguide/cppguide.html (funnet 10 05, 2020).

Graham, P. *Hackers and painters: Big Ideas From The Computer Age.* O'Reilly Media, 2004.

Grenning, Kent Beck James, et al. *Agile Manifesto.* 2001. http://www.agilemanifesto.org (funnet 08 17, 2020).

Hart, Casey, B. «Free-to-Play.» I *The Evolution and Social Impact of Video Game Economics*, av Hsuan-Yi Chou, 61-78. Lexington Books; Illustrated edition, 2017.

Hartson, Rex, og Pardha S Pyla. *The UX Book: Process and Guidelines for Ensuring a Quality User Experience 2012.* Morgan Kaufmann; 1st edition, 2012.

Highsmith, Jim. «History: The Agile Manifesto.» *AgileManifeto.* 2001. https://agilemanifesto.org/history.html (funnet 08 15, 2020).

Jørgensen, Kristine. «Games and Transgressive aesthetics.» *gta.w.uib.no.* 02 06 2015. https://gta.w.uib.no/2015/02/06/a-short-introduction-to-transgressive-aesthetics-in-games/ (funnet 02 18, 2020).

Jull, Jesper. *Handmade Pixels: Independent Video Games and the Quest for Authenticity.* MIT Press, 2019.

Kent, Emma. «EA faces class action lawsuit over loot boxes in Canada.» *Eurogamer.* 2020. https://www.eurogamer.net/articles/2020-10-22-ea-faces-class-action-lawsuit-over-loot-boxes-in-canada (funnet 10 28, 2020).

Kjøs, Skjald. «NTNUOPEN.» *ntnu.no.* Mai 2020. https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2663823/no.ntnu%3Ainspera%3A56974044%3A24751243.pdf?sequence=1&isAllowed=y (funnet 09 25, 2020).

Kozbelt, A, M Dolese, og A Soidel. «APA PsycNet.» *psycnet-apa-org.* 2012. https://psycnet-apa-org.pva.uib.no/fulltext/2011-22084-001.html (funnet 08 11, 2020).

Kozbelt, Dexter A, M Dolese, og A Seidel. «The aesthetics of software code: A quantitative exploration.» *APA psycNet.* 2012. https://psycnet-apa-org.pva.uib.no/record/2011-22084-001 (funnet 08 30, 2020).

Latour, B. «On actor-network theory: A few clarifications.» I *Soziale Welt, 47(4)*, 369-381. Nomos Verlagsgesellschaft mbH, 1996.

Lawton, Henry Shawn. «Web Content Accessibility Guidelines (WCAG) Overview.» *Web Accesibility Initiative.* 2020. https://www.w3.org/WAI/standards-guidelines/wcag/ (funnet 10 15, 2020).

—. «Web Content Accessibility Guidelines (WCAG) Overview.» *Web Accessibility Initiative.* 2018. https://www.w3.org/WAI/standards-guidelines/wcag/#wg (funnet 10 15, 2020).

Liberal Arts. «Why modern Liberal Arts?» *Liberal arts: modern liberal arts education.* 2020. https://liberalarts.online/why-modern-liberal-arts/ (funnet 10 28, 2020).

Mateas, Michael, og Montfort Nick. *A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics.* 2005. https://eis.ucsc.edu/papers/a_box_darkly.pdf (funnet 09 15, 2020).

Meisenberg, Gerhard. «Researchgate.» *Researchgate.net.* 2018. https://www.researchgate.net/publication/325665493_Editorial_The_Two_Cultures_An_Update (funnet 08 10, 2020).

Menninghaus, Winfried, et al. «What Are Aesthetic Emotions?» *Reaserchgate.net.* September 2018. https://www.researchgate.net/publication/327779286_What_Are_Aesthetic_Emotions (funnet 08 21, 2020).

Moore, David S. «Statistics among the Liberal Arts.» *Taylor & Francis Online.* 2012. https://www.tandfonline.com/doi/abs/10.1080/01621459.1998.10473786 (funnet 10 15, 2020).

Nobledesktop. «Front-end web development.» *Nobledesktop.* 2020. https://www.nobledesktop.com/image/coding-example-hipstirred.jpg (funnet 10 30, 2020).

Null, Roberta. *Universal Design: Principles and Models.* CRC Press, 2013.

Oram, A, og G Willson. *Beautiful code.* O'Reilly Media, 2007.

Orji, Rita, Gerry Chan, Ali Arya, og Zhao Zhao. «Motivational strategies and approaches for single and multi-player exergames: a social perspective.» *Researchgate.* 2019. https://www.researchgate.net/publication/337020353_Motivational_strategies_and_approaches_for_single_and_multi-player_exergames_a_social_perspective (funnet 10 30, 2020).

Parliament, EU. *Directive (EU) 2019/882 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL OF 17 APRIL 2019 ON THE ACCESSIBILITY REQUIRMENTS FOR PRODUCTS AND SERVICES.* 2019. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32019L0882 (funnet 09 24, 2020).

Preece, Jenny, Yvonne Rogers, og Helen Sharp. *Interaction Design: Beyond Human-Computer Interaction.* John Wiley & Sons Inc, 2002.

Regjeringen. *Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger.* 2013. https://lovdata.no/dokument/SF/forskrift/2013-06-21-732 (funnet 09 10, 2020).

—. «Nye læreplaner for bedre læring i fremtidens skole.» *Regjeringen.* 18 03 2019. https://www.regjeringen.no/no/aktuelt/nye-lareplaner-for-bedre-laring-i-fremtidens-skole/id2632829/ (funnet 09 25, 2020).

—. «Om forholdet mellom «funksjonshemning» og «særlig behov for opplæring, 12.2.» *Regjeringen.no.* 2020. https://www.regjeringen.no/no/dokumenter/nou-2001-22/id143931/?ch=13 (funnet 09 25, 2020).

Royce, Winston W. *Managing the development of large software systems: concepts and techniques.* 1970. https://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf (funnet 08 25, 2020).

Sack, Warren. *The Software Arts.* The MIT press, 2019.

Saffer, Dan. *Designing for Interaction: Creating Innovative Applications and Devices (Voices That Matter).* New Riders; 2nd edition, 2009.

Saffer, Dan. «Figure 1.15 The disciplines surrounding interaction design.» I *Designing for Interaction: Creating Innovative Applications and Devices*

(Voices That Matter) 2nd Edition, av Dan Saffer, 20. New Riders; 2nd edition, 2009.

Schmit, Michael. «What is Assembly.» *Sciencedirect.* 1995. https://www-sciencedirect-com.pva.uib.no/topics/engineering/machine-language (funnet 08 30, 2020).

Squarespace. *Squarespace.* 2020. https://www.squarespace.com/ (funnet 10 28, 2020).

Stack Overflow. «Is there a human readable programming language? [closed].» *Stackoverflow.* 2008. https://stackoverflow.com/questions/202750/is-there-a-human-readable-programming-language (funnet 10 05, 2020).

Stoica, Marian, Marinela Mircea, og Bogdan Ghilic-micu. «Software Development: Agile vs. Traditional.» *proquest.* 04 2013. https://search-proquest-com.pva.uib.no/docview/1492882301/fulltextPDF/A5F601E7AD9F4E48PQ/1?accountid=8579 (funnet 09 04, 2020).

Sutcliffe, Alistar G. «Design for User Engagement: Aesthetic and Attractive User Interfaces.» *researchgame.* 2019. https://www.researchgate.net/publication/220696087_Designing_for_User_Engagement_Aesthetic_and_Attractive_User_Interfaces (funnet 10 04, 2020).

Sutherland, Jeff, og Ken Schwaber. *The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework.* 2011. https://d1wqtxts1xzle7.cloudfront.net/61009942/SEHR_WICHTIG_erste_Kapitel_fur_die_theoretische_Umsetzungscrumpapers20191024-10891-p9l46b.pdf?1571965855=&response-content-disposition=inline%3B+filename%3DThe_Scrum_Papers_Nut_Bolts_and_Origins_o.pdf&Expires (funnet 11 17, 2020).

Švelch, Jaroslav. *Gaming the Iron Curtain: How Teenagers and Amateurs in Communist Czechoslovakia Claimed the Medium of Computer Games.* MIT Press, 2018.

Tauber, A I. *The Elusive Synthesis: Aesthetics and Science.* Kluwer Academic Publisher, 1996.

Theocharis, Gerogios, Marco Khurmann, Jürgen Münch, og Philipp Diebold. «Researchgate.» *Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices.* December 2015. https://www.researchgate.net/publication/281546858_Is_Water-Scrum-Fall_Reality_On_the_Use_of_Agile_and_Traditional_Development_Practices (funnet 08 27, 2020).

Theunissen, Morkel. «Software Engineering Professionalism.» *UPSpace Institutional Repository.* 2009. https://repository.up.ac.za/handle/2263/9192 (funnet 10 28, 2020).

unknown. *System development comic [figure 1].* 2020. https://i.redd.it/7dcz5131wak01.png (funnet 08 13, 2020).

WebAim Contrast Check. 2020. https://webaim.org/resources/contrastchecker/ (funnet 10 15, 2020).

Zangwill, Nick. «Aesthetic Judgment.» *Standford Encyclopedia of Philosophy.* 2019. https://plato.stanford.edu/entries/aesthetic-judgment/#JudgTastBigQues (funnet 11 05, 2020).

Appendix A) “Interviews”

These interviews were done in Norwegian and the transcribed information is found bellow.

PO 1:

Med utgangspunkt ifra denne tegneserie stripa. Hvilke av disse rutene eller områdene du har mer med å gjøre enn andre. Eller som du har mer interesse for en andre. (1:30)

“Jeg holder på med å oversette kundens krav til ønsker som utviklerne kan forstå og jobbe etter. (2:00) Å forstå hva ikke bare hva de seier, men hva de trenger.”

Hvilke av disse utfordringene finner du vanskeligst

“Det er vel kanskje det at en kunde kanskje ikke har verktøy, språk eller en bakgrunn for å klare å formidle hva den trenger. Det er kommunikasjon mellom de forskjellige fagområdene.” (03:00) “Det er kommunikasjon som er nøkkelen til alt” (3:50)

Jeg går ut ifra det skjer endringer underveis i utviklingsfasen. At det brukeren vil ha endres, hva føler du om dette og hvordan håndterer du slikt? (4:57)

“Det er en hverdagslig sak forså vidt. Det finnes ingen prosjekter uten endringer, så det venner man seg til. Det er umulig å spesifisere slik som vi gjorde I gamle dager. Det ble veldig detalj spesifisert hva som skulle bli laga, men er nesten ikke mulig fordi ting vil forandre seg […] Det å ha endringer skjer heile tida, dette som har gjort at agilt har blitt tatt veldig mye mer imot” (5:15)

Føler du av og til press grunnet tidsplanen er forliten? (16:05)

“Ja, særlig fordi at måten vi får arbeid er ved å vinne anbud som blir utlyst og så må vi konkurrere med andre selskap, og da er det ofte en urealistisk tidsplan som ligger i bunnen” (16:15)

PO 2:

Hva interesser har du innenfor jobben din. Hva er det du holder på med og liker å gjøre? (5:00)

“Det har jeg faktisk reflektert over de siste par årene også fordi at dette er mitt fjerde yrke. […] Jeg er en person som elsker å lære nye ting og som nummer to på listen så liker jeg å løse problemer, og det får jeg brukt veldig godt i denne jobben. Kombinere utforsking av brukere sine utfordringer, problemer og hverdag med både finne og beskrive løsninger for di. Men også de å hjelpe utviklere å grave i problemstillinger […] Særlig dette med problemløsningen er topp for meg.” (5:29)

Så, når du snakker med kunden, føler du at de ikke veit hvordan utvikling, at de har et gammelt tankesyn på hvordan de fungere? At de tror at de fungerer som fossefall? (21:00)

“Ja jeg tror en del ligger der at vi ikke har innsikt i denne måten å jobbe på. Men jeg tror det at, de fremdeles driver gammeldags program ledelse, med då, leveranse av en hel del systemer. Og da tenker man fremdeles, her er alt vi trenger, her er utviklingsfasen og her er test fasen og leveranse til slutt. Dette fordi det lar seg planlegge, telle og måle. Det passer mye bedre på et telle ark men gir ikke beste svar og løsninger.” (21:20)

Har du noe tanker rundt, synes du at tanke gangen må endres slikt at kunden får en bedre forståelse for hvordan disse prosessen fungere på utvikler siden? (22:35)

“Jeg er litt splittet, og grunnen er at jeg er et regne ark menneske. Ting er veldig lett når det passe i et regneark. Og så har jeg igjennom denne jobben lært at det er en måte å jobbe på som jeg forstår intellektuelt at det gir bedre resultater, det vi ender opp med for brukeren. Men er fortsatt vanskelig å få de til å matche inni hodet mitt” (22:50)

UX 1:

Hva er det du liker med UX?

“Det er veldig gøy å løse problemer. Å lage et verktøy som folk trenger og bruker. Noe di får en verdi av” (1:50)

Hvis du har lagd deg et bilde over hvordan et program skal fungere og så kommer det en endring så gjør at denne løsningen du mener var god må endres. Hva gjør dette med deg? (17:55)

“Nei det er jo selvfølgelig kjedelig når man har en veldig god ide og er sikker på det blir fenomenalt, men er alltid noe som legger kjepper i hjulene på en måte som gjør at det ikke kan gå den veien. Det er jo synd, men man må jo alltid omstille seg da, det er jo veldig ofte dette skjer i stor og liten grad.” (18:13)

Du nevnte intuisjon, er det då at du titter på løsningen og ja detter ser ut som det kommer til å virke på grunn av at det føles rett? Og kan du forklare litt hva som er grunnen til at intuisjonen din kan ta disse beslutningene? (21:10)

“For meg så er det jo, når jeg jobber for meg selv så er det veldig viktig å følge mageføleIsen. Ofte er det jo relativt rett det jeg gjør, eller det jeg går for. Og det er mer hvor kjapt jeg ser hva som føles riktig. Uten å gå fordyp i det å analysere alle detaljer. Det er ikke viktig når jeg jobber for meg selv, men mer viktig når vi tester med brukere." **Er de det at det bare ser rett ut eller er det at det er vakkert?** “Ja, det føles veldig rett ut, men det går utpå hva jeg jobber med eller ser på. Informasjonsstruktur eller rent GUI, hva som ser bra ut, er øyenfallende eller en Call to action. Det blir ofte veldig detalj orientert, endre en farge her så blir det bedre. Det er mye frem og tilbake.” (21:30)

(23:40) Jeg har et spørsmål angående code review, men dere har noe tilsvarende innen UX. Er dette noe du er ofte borti?

"Ja, det er noe vi gjør ofte inni SCRUM teamene. Nå har starter å gjøre det på tvers av avdelingene også. Sånn at vi kan dele litt ider med andre prosjektledere eller andre team. Men innad i hvert SCRUM team har vi gjerne en review av hva som skal lages i neste sprint. Og da har vi en diskusjon med utviklerne om det er mulig å kode dette her, eller finnes det teknisk bedre løsninger. " **Sitter dere ofte på team med utviklere?** Ja, vi kommunisere ofte med utvikleren og på kontoret så sitter vi sammen i teamet. Så de er veldig mye lettere å diskutere med hverandre å stille kjappe spørsmål (23:55)

Hva tanker har du rundt universell utforming? (39:50)

"Det er jo veldig viktig selvfølgelig. Det har jo en veldig stor verdi for mange brukere, men det har veldig stor verdi for hva jeg produsere også. Fordi måten universell utforming fungere er at det skal fungere best mulig for alle. Og det vil jo si at UI og UX må være veldig enkelt og universell utforma. Dette får meg til å virkelig forenkle brukergrensesnittet og brukerflyten. Så uansett om man tenker eller ikke tenker på blinde skal bruke det så gjør det de mye lettere å forme produktet." (40:00)

UX 2:

Føler du at det har hjulpet at utvikling har blitt mer agile? (4:57)

“Ja uten tvil, i starten så var det veldig budsjett drevent med fossefall, man skal gjøre det ene foran det andre. Man har fått en større forståelse i de forskjellige organisasjonene at det er på lang sikt lurer å jobbe mer smidig, men det koster mer. Hvis man jobber etter fossefall så kan man vise regningen dag en. Det kan man ikke med smidig.” (5:01)

“Offentlige kunder har jo fortsatt sine budsjetter sant, og de må holde seg til disse. Det gjør det vanskelig å jobbe veldig smidig. Det er noe med måten å tenke penger på burde vært mer smidig.” (5:40)

Du har vel gått innpå ei nettside som funker på en slik måte at du vil dra deg i håret. Er dette noe som får deg til å ville passe på at det dere lager holder en standard som forhindrer dette? (28:00)

"Sånne ting er jo med på å påvirke motivasjonen for faget, man ønsker å lage noe som hjelper mennesker og som er brukervennlighet, og på mange måte ønsker man å være noen som gjøre dette vellykket. Man ser kanskje i starten at det var veldig at man ønsket å være bedre enn de fleste. Det som er en slags reaility check etter noen år er jo at man kan være så grundig og ha fikset ting så mye man vil og tro at man har gjort alt riktig. Men med en gang man … Ting forandrer seg jo alltid også, bruksmønstre forandrer seg, folk forandrer seg. Måten å bruke mobilen forandrer seg. Scrolling og tasting, måten å gjøre ting på. Ting forandrer seg og neste

ingenting er likt ifra år til år og da blir det litt sånn at man ønsker ja, men man ønsker kanskje bare å være de som føller trender og holder det jevnt og gående. Man ønsker selvsagt ikke å være de som lager noe som får deg til å dra deg i håret, men det er en hårfin balanse mellom å prøve å være innovativt også. Veldig ofte er det tingene som ikke er innarbeidet, det kan være bedre løsninger, men er ikke innarbeidet så det oppleves som dårligere. Det er noe med å vite når man skal være først ut med noe."(28:15)

Måten dere utvikler UX på, liker du måten dere gjør dette på? (32:30)

“Man starter med innsiktsarbeid, lager use case og personas, finner ut hva disse trenger for å nå sine mål, og om det samsvarer med forretningsmålet til bedriften. Prøve å finne ut vist et av målene til bedriften er selvhjulpen, de vil få ned henvendelsene til service disken. Så trengs gode selvhjelp sider der designet er brukervennlig. Man tar på seg brukerens, setter seg i brukerens sko og prøver å finne ut hvordan jeg vil gjøre dette, fylle ut dette skjema sjøl hvordan skal jeg klare dette etc. Og da er det jo sånn at universell utvikling er veldig viktig […] så er det klart at man må lage knapper, koden god nok til skjermleser, godt språk og annet.” (32:43)

Ifra det du nå nevnte, hva er det du liker best med det som nett ble nemnd. (34:40)

“Ja, det aller kjekkeste for min del er å jobbe med bruker testing, når man allerede har lagd en prototype av et konsept og se hvordan brukeren reagere på nye konsept og det som er blitt lagd” (34:50)

Programmer 1:

“Når jeg begynte så var det veldig vanlig med fossefall, vi fikk en stor bibel fra kunden, dette er hva vi vil ha. Første jeg var utenfor var 1400 sider med beskrivelse om hvordan produktet skulle være. Noe som betydde at når jeg kom in så var det satt av 3 måneder for å forstå hva de ville ha. Og da satt alle sammen i tre måneder for å forstå hva de ville ha. Fra kundens ståsted så var de ferdig.” (5:30)

Føler du at det er lettere vist prosjekt eier har mer IT kunnskap? (26:06)

“Det er ikke tvil at de gangene jeg har hat en prosjekteier med dyp teknisk forståelse, at jeg har funnet det lettere. Men samtidig, eller, trenger ikke være dyp teknisk forståelse, men tema. En av prosjekteierne vi har er utdannet om hva vi utviklere nå, har vært vel så viktig som en annen prosjekteier vi hadde før […] Personlig for meg så har jeg funnet det vanskeligere, men for andre så har di de lettere” (26:17)

Holder du på med code reviews i måten du jobber på? (39:40)

"I bedriften er jeg koordinator for frontend guildet. Det betyr at de forskjellige teamsene har folk med interesse for forskjellige ting. De kan da bli med i guildet for å få delt erfaring fra gamle utviklere til nye. I frontend guildet har ansvar for mange av de interne bibliotekene som er blitt lagd. Og de betyr at man får et par personer som gatekeepers. Typisk for alle teams er interne code reviews før merge request etterfulgt av en code review. Dette skjer også i front end guildet." (39:47)

Når du ser på kode, finner du av og til biter eller alt vakkert? I så fall, hva finner du vakkert med den? (43:30)

"Ja absolutt, det er mange forskjellige ting jeg finner vakkert med kode. I sinn tid når ruby on rails dukket opp i 2005. Så såg jeg den lille screencasten, 15 minutter lag en blogg ting. Og to ting som jeg la merket til. En var språket RUBY. Det var fryktelig vakkert, måten det var lagd på, men også at det var som engelsk. Mye god Ruby kode er som engelsk, man kan ta funksjonsnavn gi til noen som ikke kan programmering og de kan forstå hva som skjer. Den andre tingen var tekst editoren. Det førte til at jeg ville gjøre ruby on rails etter dette. De er fortsatt favoritt språket mitt fordi jeg syns koden er flott å se på. Andre ting som kan være fint er flotte måter ting er løst på. Ikke nødvendig vis smarte, jeg er veldig lite begeistret for smarte fordi de ikke er lesbar. Man trenger de av og til, men i bedriften jeg nå jobber i så finnes det masse eksempler på at funksjoner finnes i en lesbar og en smart utgave. Noen steder vet vi at vist en variabel er over ett spesifikt tall så må vi bruke den smarte.

Men vi vil ikke at den smarte skal være den folk jobber med. De skal kunne forholde seg til den lesbare […] Jeg mener ikke lesbar kode er til stor grad dårlig kode, med språkene vi har i dag og de maskinene vi har så er det sjeldent at ikke lesbar kode er bedre enn lesbar kode."

Det har jo komt ett nytt regelverk rundt universell utforming, har du noen tanker rundt universell utforming? (54:45)

"Jeg er jo veldig mye større tilhenger av universell design en det motsatte. Alt for ofte blir ting lagd, designet og så kommer tilgjengeligheten oppå der igjen. Et eksempel jeg liker om dette er butikken som oppdaget etter at de hadde åpnet at rullestoler kom seg ikke inn. Men vi har en dør på siden og så kan vi lage ei rampe opp dit. Eller hvis man bare tenkte universelt med en gang så hadde det aldri vært en trapp ved hovedinngangen. Ifra de firmaene jeg har jobbet for så er det ikke tvil at denne tankegangen er veldig kostnads besparende, men endrer også hvordan man designer og tenker igjennom alt. […] Jeg er en veldig stor fan av universell design og har prøvd å få UX til å bevege seg i samme retning som universell design gjør."(55:00)

Programmering / programvareutvikling er et stort fagfelt, ser du på det som matte ingeniør, eller er det mer kunst og litteratur? (58:40)

“Samtidig når man ser at ingen skriver kode likt […] Man ser en stor grad med folk med språk utdanning som gjør det godt innenfor programmering, som gjerne ikke har en stor teknisk matte naturfag bakgrunn. Til syvende og sist så tror jeg at mye handler om kommunikasjon, mye av programmering handler om å skrive og hvis du er å flink til å skrive så kan du klare deg godt innenfor programmering. Det er slett ikke alle som må holder på med dyp teknisk kode innenfor AI eller tilsvarende.” (59:33)