

https://dspace.org/

Investigating the software artwork creation process from an Agile perspective.

Grotepass, Maria

2013-01-14
http://hdl.handle.net/10539/12270

Downloaded from a DSpace repository. https://dspace.org/

Investigating the Software artwork creation process from an Agile perspective.

A Master thesis submitted to the University of the Witwatersrand School of Arts, Digital Division. In partial fulfilment of the requirements for the Degree of Masters of Art by Dissertation.

Student:	Maia Grotepass
Student ID:	486105
Supervised by:	Tegan Bristow
	Stephen Lewitt

Johannesburg, March 2012

Declaration Page

I declare that this is my own unaided work. It is submitted for the degree of Master of Arts at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree of examination in any other university.

Maia Grotepass

March 2012

Table of Contents

1 INTRODUCTION **1**

1.1 Research questions 5

1.2 Methodology 5

1.3 Overview 8

2 CREATIVITY AS PROCESS **10**

2.1 Definition of creativity and creative process 10

2.2 Overview of the models for the creative process 12

2.3 Cognitive skills required for creative activity 17

2.4 Research on art and technology collaboration 20

2.5 Characteristics of tools and process which assists creativity 22

2.6 Conclusion 26

3 SOFTWARE DEVELOPMENT PROCESS **27**

3.1 Different Methodologies 30

3.2 Agile values applied to software art 31

3.3 Agile principles applied to software art 32

3.4 Stakeholder roles in digital art projects 36

3.5 Agile process phases applied to software art 39

3.6 Agile development practices relevant to digital art production 41

3.6.1 Iterative process *41*

3.6.2 Regular Feedback with stakeholders *42*

3.6.3 Continuous testing *43*

3.6.4 Collaboration *45*

3.7 Additional development practices relevant to digital art production 46

3.7.1 Configuration Management *46*

3.7.2 Testing *48*

3.8 Conclusion 50

4 SOFTWARE AS MEDIUM **51**

4.1 Software 51

4.2 Medium 52

4.3 Software as material or technique 55

4.3.1 Characteristics of software as medium *55*

4.3.2 Programmable *58*

4.3.3 Non-static, active *59*

4.3.4 Running algorithms *60*

4.3.5 Mutable, flexible *60*

4.3.6 Re-use, re-mix *62*

4.3.7 Software as tool *63*

4.4 Software as transmission medium 64

4.4.1 Software art relates to conceptual art *64*

4.4.2 Software and language *68*

4.4.3 Comparing code and language *69*

4.4.4 Software as unconscious of language *70*

4.5 Software as functioning environment 71

4.5.1 Code and machine as model of reality *71*

4.5.2 Software as muse 72
4.5.3 Software as metaphor 73
4.6 Software as end product 74

5 DISCUSSION 76
5.1 Research questions and how they relate to the areas of study 76
5.2 Interviews with Artists 77
5.2.1 Brogan Bunt 78
5.2.2 Joshua Goldberg 80
5.2.3 Pierre Proske 83
5.2.4 Nathaniel Stern 89
5.2.5 Pall Thayer 89
5.2.6 Comparison of interviews 95
5.2.7 Opinions of artists and technologists 97
5.3 Software as medium, the creative process and interview results 98
5.4 How do practising software artist experience their development process? 101
5.5 The Agile process and interview results 101
5.6 The Agile process and the creative process 106
5.6.1 Creative process models and Agile process compared 106
5.6.2 Iteration 107
5.6.3 Collaboration 108
5.6.4 Flexible, simple and adaptive 111
5.7 How does this process compare with the Agile software development process? 112
5.8 Engineering process and software artists 113
5.8.1 Agile process elements that support software artists 113
5.8.2 Engineering process elements that support software artists 114
5.8.3 Engineering process elements to avoid 116
5.9 Creative process elements support coders 117
5.10 Process is not the whole picture 118

6 PRACTICAL WORK 120
6.1 Installation description 120
6.1.1 commentCompile 124
6.1.2 commitOften 126
6.1.3 initBefore 128
6.1.4 interfaceInstead 130
6.2 Artistic choices 135
6.2.1 Spaces and screens 135
6.2.2 Interaction strategies 140
6.2.3 Colour 143
6.2.4 Font 147
6.3 Technical implementation choices 148
6.3.1 Tools and hardware 148
6.3.2 Development process 149
6.3.3 Architecture 153
6.3.4 Sound 155
6.3.5 Testing 155
6.3.6 Reliability requirements 159
6.3.7 Installation/deployment 160

7 CONCLUSION 162

8 APPENDICES I

9 LIST OF WORKS CITED XX

List of Figures

Fig. 1: Four Phase Creative process model 14
Fig. 2: Geneplore iterative creative process model 15
Fig. 3: Runco and Chand Iterative creative process model 16
Fig. 4: Simplified Waterfall development model 27
Fig. 5: Simplified Agile development model 28
Fig. 6: Digital artwork team member roles 37
Fig. 7: Agile process phases applied to software art 39
Fig. 8: Map of chapter and section relationships 76
Fig. 9: Image 5 and 6 of the series Loom Bunt (2011) 80
Fig. 10: Image 14 and 15 of the series Loom Bunt(2011) 80
Fig. 11: User interface of the program Dervish Goldberg (2002) 82
Fig. 12: Frame from the video dervish meditation - 2012 Goldberg (2012) 83
Fig. 13: Screenshot of the video dervish mediation - 2012 Goldberg (2012) 83
Fig. 14: Screenshot from Abstract Microecologies Proske (2006) 87
Fig. 15: Screenshot of a video, Abstract Microecologies Proske (2006) 88
Fig 16: Screenshot of video, Abstract Microecologies Proske (2006) 88
Fig. 17: Microcodes website Pall Thayer (2009 - 2012) 91
Fig. 18: an Icelandic Landscape, code and output Pall Thayer (2009) 92
Fig. 19: white on white, code and output Pall Thayer (2009) 93
Fig. 20: cnn Dada, code and output Pall Thayer (2009) 94
Fig. 21: commentCompile screenshot 124
Fig. 22: commitOften screenshot 126
Fig. 23: initBefore screenshot 128
Fig. 24: interfaceInstead screenshot 130
Fig. 25: Mark making investigation 133
Fig. 26: Mark making investigation 133
Fig. 27: Mark making and layering investigation 134
Fig. 28: Floor-plan showing installation spaces and screen placement 135
Fig. 29: Blue Screen of Death 145
Fig. 30: Web-cam M Grotepass 146
Fig. 31: Colour, shape and movement choices 146
Fig. 32: Goal and exploratory approach in workbook 151
Fig. 33: Goal and exploratory approach in workbook 152
Fig. 34: Investigation of possible visual handling 152
Fig. 35: Screen material test 158
Fig. 36: initBefore memory footprint graph 158
Fig. 37: commentCompile memory footprint graph 159
Fig. 38: commitOften memory footprint graph 159

List of Tables

Table 1: Summary of cognitive styles used by Candy and Edmonds 21
Table 2: Digital artwork team stakeholder roles 37

Appendices

Appendix I: Contents of disk I
Appendix II: Extract of DataMote particle class II
Appendix III: Email interview with Brogan Bunt III
Appendix IV: Transcription of Skype Interview with Joshua Goldberg V
Appendix V: Skype interview with Pierre Proske XI
Appendix VI: Email interview with Nathaniel Stern XVI
Appendix VII: Email interview with Pall Thayer XVII

1 Introduction

Technology is ubiquitous and integrated into our society. People have access to cell phones and computers. The Internet connects people and makes ideas and information available to people using technology. Technology is visible in popular culture, on television and in the media. The media are presented to people using technology, television and film. Technology mediated communication has become widespread. Artists respond to the technology around them. In addition artists use technology as medium to express ideas, using the same communication mechanism to capture and communicate their ideas. Using technology as medium allows the artists to explore the inner-workings of areas of technology. This gives the artist direct experience, which allows for a critical response to the effects of technology on society and communication.

Artists choose code because it is an active medium. The code is executed by the machine and elements of the artwork changes. It is non-static. Software as medium facilitates the re-use and sharing of code among practitioners. The code can be copied and modified without changing the original version. Time-consuming drawing or painting actions can be executed by the computer and so aspects of creating an artwork can be automated allowing the artist to explore more variations of the process in a shorter time. Software as medium facilitates interactive artwork because the artist can code the artwork to respond differently to viewer interactions and because the technology is built to take input from viewer-players as well as other computer systems. These concepts: active, re-usable, supports

automation, interactive, are examples of characteristics of the medium, which artists can use in their creative process.

Participants in the field of digital art who create code based artworks, approach the field from at least two directions. Coders become artists. Artists become coders. When coders become artist and/or artist choose code as medium, the software development process and the creative process happen at the same time. Two worlds collide. The medium influences the creative process. The goals of the artist influence the software development process. What is unique and different to the two approaches to the process? Why is it useful to understand this? There are elements of the software development process that can be used without hampering the creative process. Likewise there are practices which support creativity, which can help software projects to accommodate ambiguity and result in innovative results. To understand what they are, requires an understanding of both the creative process and the software development process. An understanding of both processes will offer insights into the different approaches which can aid collaboration.

This dissertation argues that there are elements of the Agile engineering process that can help artists who use code as medium. Code is different to other art media because it requires interaction with a computer. It requires translating ideas into a form which can be executed; a precise set of instructions understandable to a machine. Using code, multiple versions can be created easily. Multiple versions can cause confusion but can also supply multiple solutions to a problem and a mechanism to assist exploration. The Agile engineering process helps to manage multiple versions and helps to manage the code creation process.

Conversely, an understanding of the creative process can help engineering processes to be more flexible and produce creative solutions. Case studies document that collaboration between technical people and artists highlight a difference in approach. The expectation of the results of the process is one of the ways in which the approaches differ. The expectations are a clear goal on the one hand and an exploration and discovery on the other hand. If the goal is narrowed down at the start of the project and the playful experimental approach is avoided, some solutions may never be explored.

The artistic creative process has been documented and investigated. It is an illusive field as there are as many variations in process as there are artists. In addition it is not easy to pinpoint the factors that affect the success or failure of the artistic process. Some artists are consciously or sub-consciously exploring the edges of their medium and process in an attempt to create something that has not been made before, something unexpected and unpredictable. Results, which from a software industry point of view could be seen as an error, a failure or sub-optimal behaviour, can create something unusual. To understand the creative process a cognitive psychological approach was chosen as a starting point because software development is a cognitive activity. An attempt was made to choose a useful framework to investigate creativity as process.

The software development process has been investigated and documented by the engineering discipline. The process has been modelled using various approaches. Initially the Waterfall model was created which described software development as a series of steps one following upon another. This approach did not allow deviation from the initial goal which was set at the start of the project. Oth-

er ways to model the process have been proposed. For this dissertation I have chosen to look at the Agile software development process because it is useful for projects where the exact definition of what has to be created does not exist, such as digital art projects (Abrahamsson et al. 14). The Agile process is a software development process which is used in the commercial development of software products. It is particularly suitable to projects where the requirements are unknown or changing.

The Agile process is not a single methodology but rather a family of methodologies based on a set of principles. The Agile methodology is flexible in that it accommodates multiple modes of practice. It is a process model so it does not give specific instructions on the nature of the outcomes but rather suggests process steps to meet varied goals. The process steps may be applied to any kind of project but Agile is typically used for software development projects. The process suggestions provided by Agile can be adjusted as the project requires. One of the key values of the Agile model is that it remains adaptable to the needs of the project. This adaptability allows it to be used when requirements change rapidly. This may also allow the process to be adapted so that it is useful for software art projects.

The Agile iterative approach proposes that the following process phases occur: planning, implementation and testing. What makes the Agile approach different to its predecessors is that the different phases happen regularly. The planning phase is a reflection phase, the implementation phase is a making phase and the testing phase is an exploratory phase. A process model which is suitable for software development, which can accommodate reflection, making and exploration

and has adaptability as a core guideline may be a useful tool for the software artist.

1.1 Research questions

The research questions are:

How do practising software artist experience their development process? How does this process compare with the Agile software development process? How can conclusions made from a comparison between the Agile process and the discussions held with practising software artists shed light on the areas where the Agile process can assist artists and areas which might be avoided?

1.2 Methodology

This research looks specifically at artists who use software as creative medium. This group was chosen, as people in this category will have experience in the creative process because they are actively creating artwork. This group will also have experience in software development processes because their medium is software code.

A community of coders/artists were approached to record their experience of their process. The community is the openframeworks community, which was chosen as openframeworks is a collection of code that is suited to creating artwork ("openFrameworks"). It consists of a collection of c++ code and the source code is open. This allows artists to get started creating an artwork by modifying existing pieces of code but also allows access to the code so the platform won't hinder the creative process. The views of the participants were gathered by email, forum discussions and questionnaires. The questionnaires were not successful as,

in retrospect, the questions were too long and complex and participants did not respond at all. In a second attempt, direct discussion on the openframeworks forum were initiated. This approach delivered some results, but again the response was sparse. A new list of focus artists were chosen including artists who responded directly on the forum. An email with three simple questions were sent to this list of artists. Direct responses to the questions were received from all emails and a Skype interview was conducted with two of the artists. To supplement these responses, opinions published by artists in the form of blog posts and archived online discussions were gathered on the Internet. The artists were chosen because they had exhibited artwork which involved writing software in some form so they were directly involved in the coding process and the creative process. Since Agile is a process model, the artists were also chosen because they had experience in the software coding process.

Correspondence was collected from the following artists:

Brogan Bunt is a media artist and an academic. He creates video and software art work. He is the Head of the School of Art and Design, University of Wollongong, NSW, Australia. He has also written about "aesthetic issues emerging from the contemporary effort to position programming as a form of artistic practice" ("About | brogan bunt"). His work includes interactive software based work but also custom software tools. An email conversation was conducted with Bunt. Bunt provided additional documentation and symposium papers which clarified his views.

Joshua Goldberg is a New York based artist who uses software systems to perform custom sound visualisations. He calls himself a 'live visualist' ("joshuagold-

berg"). He also writes custom software tools which he has released to the public. An audio conversation with Goldberg was recorded and transcribed.

Pierre Proske is an Australian artist who merges his parallel interest in technology and the arts. He studied Electrical engineering and liberal arts at undergraduate level and completed a Masters in Art and Technology at Chalmers university in Sweden. He has worked as a sound designer and electronic musician and has exhibited and performed in Australia, Sweden, Canada, Iceland, Brazil, Japan, Austria and the Netherlands (Proske). An audio conversation with Proske was recorded and transcribed.

Nathaniel Stern is an installation, video and internet artist. He is based in USA and South Africa. He uses both traditional media and technology to create his work. He teaches at the Department of Art and Design at the University of Wisconsin and writes about art (Stern, "nathaniel stern : short artist biography"). An email conversation was conducted with Stern. Artists statement and discussion of his work was used to support this email.

Pall Thayer is an artist and lecturer based in Iceland. He created a series of small code based artworks where the meanings are deciphered from a combination of the title of the work, reading the code and running the code (Thayer, "pall thayer bio and interview"). An email conversation was conducted with Thayer. Documentation of the conceptual background of his work was used to extend his views.

1.3 Overview

Chapter 2 will focus on creativity as process. The definitions of creativity will be investigated. This chapter will explain why the cognitive psychological approach is chosen for this study. It will then outline important aspects of the creative process that needs to be maintained by any process which is adopted in a project.

Chapter 3 looks at the software development process. It will give the reasons why the Agile process is chosen as framework to investigate software art projects. It will discuss the values and principles on which Agile processes are based. It will discuss aspects of the development process that may be useful for software development from an artwork development point of view. It will explain which areas of the Agile process is not applicable to artwork development.

Chapter 4 will look at software as artistic medium. It will discuss what it means when we use the term medium in this context. It will elaborate the reason artists choose to use software as medium. From a selection of artist opinions a subset of characteristics of the medium will be defined. These characteristics will be discussed with relation to academic writing on the subject by Manovich and Hayles.

Chapter 5 presents a summary of interviews which were conducted with five artist/programmers. Connections between their views and concepts presented in the preceding three chapters will be made and discussed with the aim to show overlaps and differences in the respective fields. These overlaps and differences identified will be used to answer the research questions.

Chapter 6 documents my artistic practice for this research which culminated in an interactive installation exhibition titled *nullPointerException*. This chapter dis-

cusses the works that make up the exhibition. It also relates the subject matter of the practical installation to the research. It records the choices of aesthetic and technical elements of the work. It elaborates on the reasoning behind the choices. It looks at the creative and development process which I used when creating these practical works and documents the connections with the research in the preceding chapters. I also include a disk with video documentation of the exhibition, website, catalogue and source code of the work. The contents of the disk is listed in Appendix I: Contents of disk.

The creative portion of my dissertation includes the following works:

- *commentCompile* (2011): An interactive projected installation work, coded in C++ using the Kinect as sensing device. This work uses the coding phase of software development as starting point.
- *commitOften* (2011): An interactive projected installation work, coded in C++ using the Kinect as sensing device. This work uses source code control and change management as starting point.
- *initBefore (*2011): An interactive projected installation work, coded in C++ using the Kinect as sensing device. This work uses algorithms and memory views of running software as starting point.
- *interfaceInstead* (2011): An interactive projected installation work, coded in C++ using the Kinect as sensing device. This work uses the design phase and the difference of human representation and machine representation of software architectural elements as starting point.

2 Creativity as process

Whatever medium artists choose, they are involved in a creative process. If software is the medium, the process is also a software development process. Software development is a cognitive activity. So in this chapter the creative process is investigated from a cognitive point of view. The cognitive psychological approach studies mental processes, such as remembering and learning. The focus is on internal mental processes, which aren't necessarily observable as behaviours. Looking at the research on creativity as a cognitive process provides a way to understand software development process from a creative process point of view. Understanding the creativity as a process will help to understand where a software process such as the Agile method can help or hinder the creative process. This will allow identification of elements of the process which needs to be maintained and not disrupted.

As a first step different definitions of creativity will be compared. Secondly models of the creative process will be discussed. To follow, different cognitive skills, abilities and cognitive approaches to creativity will be investigated. Lastly recommendations for tools and processes that support creativity will be summarised.

2.1 Definition of creativity and creative process

Different authors and researchers have different definitions of creativity. A source that covers multiple aspects of the creative process is the book, *Creativity: Understanding Innovation in Problem Solving, Science, Invention and the Arts* by R.

Weisberg (2006). Weisberg defines creativity as "Creative thinking occurs when a person intentionally produces a novel product while working on some task" (70). Lubart defines the creative process as "the sequence of thoughts and actions that leads to a novel, adaptive production" (295). Buss quotes Mel Rhodes with the definition, "The word creativity is a noun naming the phenomenon in which a person communicates a new concept (which is the product). Mental activity (or mental process) is implicit in the definition, and of course no one could conceive of a person living in a vacuum" (17). Sternberg rephrases "Creativity refers to the potential to produce novel ideas that are task-appropriate and high in quality" (360).

The definitions compare in that they describe the production of something novel as a key element defining creativity. Some definitions do not regard an act as creative if the culture within which it is created does not regard it as novel (Csikszenthihalyi 313). In addition an act is not regarded as creative if it happens by accident, that means it has to be an intentional act (Weisberg 66).

Dietrich warns that endeavours which investigate creativity should not fall into the trap of viewing creativity as monolithic. It is a complex activity consisting of multiple cognitive activities and influenced by multiple factors (Dietrich 24). Pope captures the multiplicity of creativity in the sentence "Creativity is extra/ordinary, original and fitting, ful-filling, in (ter)ventive, co-operative, un/consious, fe<>male, re...creation." This is not so much a definition but an encapsulation of different approaches to and aspects of creativity. He is of the opinion that attempting a definition is moot but that the focus should rather be on the effects of creativity or perhaps the historical and social framing of creativity (Pope 52).

There is an inherent paradox in studying creativity: "How is it ever possible to conceive of a truly creative idea? If you could anticipate the idea, it would be determined and not creative" (Hausman 14). Finke, Ward and Smith deals with this paradox by assuming that the cognitive elements which are explored have emergent qualities. That is to say the cognitive elements can give rise to ideas which have details which were not anticipated and only discovered after initial ideas are explored. The emergent quality of a cognitive element does not guarantee that the result will be creative but increases the likelihood of a creative result (Finke, Ward, and Smith 8). The experimental approach to creativity would accommodate creative ideas to be discovered.

Although all the definitions of creativity concur in that something novel is created, the creative process is difficult to pin down. It has been investigated using different methods ranging from autobiographical self reports to laboratory experiments. Each method has it's own strengths and weaknesses (Weisberg 73). Since software is a cognitive activity the focus will be on the cognitive psychological approach to investigate the creative process. From these definitions, for software artwork creation, it is important that a software development process does not hamper complex cognitive activities. It also needs to support activities which are emergent so that novel solutions can develop.

2.2 Overview of the models for the creative process

As part of the Laboratoire Cognition et Développement, Todd Lubart looked at the development of the research on creative process over the past century in his paper titled *Models of the Creative Process: Past, Present and Future* (2001). It

provides a useful overview of the models of the creative process and the changes in views from 1926 to the current models (Lubart 296). The classic model is the Four-Stage model which consists of: (a) preparation, (b) incubation, (c) illumination and (d) verification. See fig. 1. Although this model has earlier history, it is discussed by Guilford in the 1950 presidential address to the American Psychological Association (Guilford 41). Guilford's aim was to find ways to discover "creative promise" in children and to promote "development of creative personalities" (Guilford 34). Guildford finds the 4 stage model superficial because it doesn't give information on the cognitive activities of each stage (41). The four stage model is still in use with some variations and has been extended to differentiate between the problem-finding and problem-formulation phases (Lubart 297). Views that the stages are not as defined and may occur simultaneously are the main criticisms against this model indicating the need to revise or replace this model. More recent studies have focused on the sub-processes of creative activities rather than stages, creating a more dynamic model where the sub-processes can be revisited and cycling can occur between the sub-processes (Lubart 299).

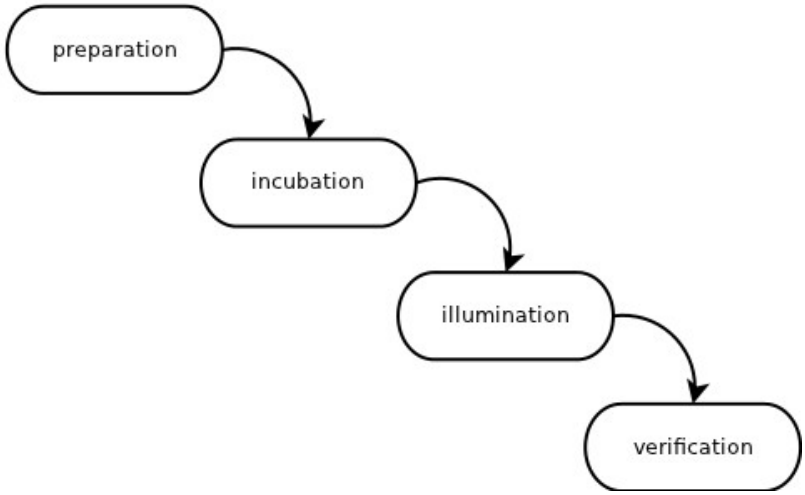


Fig. 1: Four Phase Creative process model

There are multiple studies on the sub-processes and multiple authors propose a dynamic model of the process of creativity. An example of the dynamic model and sub-process focus is the work done by Finke, Ward and Smith in the book titled *Creative Cognition: Theory Research and Applications* (1992). The goal of Finke, Ward and Smith's research is to investigate the cognitive processes and structures involved in the creative process (4). They focus on two groups of creative sub-processes: generative and exploratory processes. In this model the process oscillates between the two sets of processes, combining them in a cyclic fashion. See fig. 2. At any stage the current state of the creative product can be verified and constraints applied which will either result in the resuming of the processes or the culmination of the process (Finke, Ward, and Smith 18). This model is called the Geneplore model. The generative phase can include activities such as knowledge retrieval, idea association, synthesis, transformation and analogical transfer. The exploratory phase can include activities such as examination,

elaboration, testing of the preinventive structures, hypothesis testing and searching for limitations (Lubart 300).

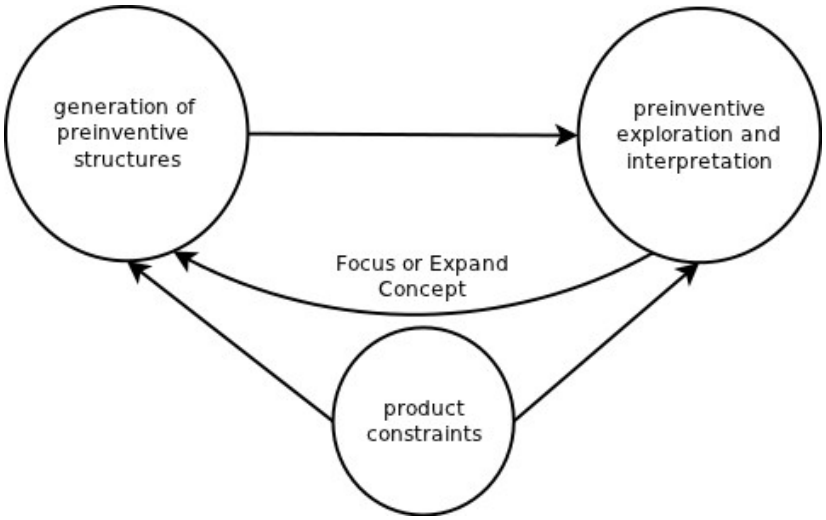


Fig. 2: Geneplore iterative creative process model

Source: Finke, Ward, and Smith Creative Cognition: Theory, Research , and Applications (M.I.T.Press:1992) 18. Print

There are other studies that separate the creative process into similar phases as the Geneplore model. One of these studies is described in the article, titled *Cognition and creativity*, by Runco and Chand (1995). Runco and Chand build on this model (252). They include problem finding as one of the phases and also include ideation, which is similar to the generative phase of Geneplore, and evaluation, which corresponds to the exploratory phase of Geneplore. Problem finding, ideation and evaluation form the primary tier of their two tier model of creativity. See fig. 3. The problem finding phase is added so that the model describes a creative process and not purely a problem solving process. The secondary tier contains contributing factors rather than controlling factors of the process such as the relationship to knowledge and the motivation. The ideation/generative phase is associated with divergent thinking. The evaluative/exploratory phase is seen as

convergent. Runco and Chand place as much value in the evaluative/exploratory phase as the problem finding and ideation/generative phase. Runco and Chand also propose that their model is not a static one but that there are interactions among the process and that the model can be recursive (Runco and Chand 245-262). So in both Finke, Ward and Smith and Runco and Chand's model we have the concept or potential of iteration and the inclusion of both divergent and convergent phases.

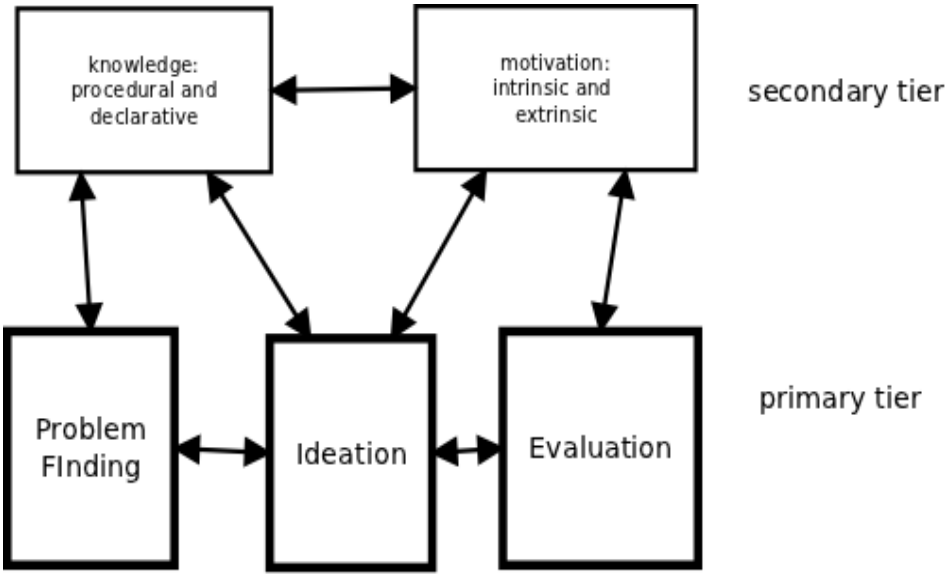


Fig. 3: Runco and Chand Iterative creative process model

Source: M.Runco and I. Chand, Cognition and creativity (Educational psychology review: 1995) 245. Web

Looking at the models of the creative process the four stage model is a starting point, but for software artwork development a model which supports iteration and which supports different phases is useful. Two phases in the Geneplore and Runco and Chand's model are identified because the two phases describe very different cognitive activities. The one phase is the generative (in Finke, Ward and Smith's model)/ideation (in Runco and Chand's model) phase. In this phase multiple possible ideas are generated. The other phase is the exploratory (in Finke,

Ward and Smith's model)/evaluation (in Runco and Chand's model) phase. In this phase ideas are discarded because the ideas are tested for suitability and the solution is refined. A software development process must be able so support both of these very different phases.

2.3 Cognitive skills required for creative activity

Creating something novel implies something that is not the same as that which came before, a "break with the past" (Weisberg 101). As a result many studies of creativity postulated "extraordinary thought processes" (Weisberg 101). However Weisberg is of the opinion that both experience and creativity is required to create something and that the same cognitive processes are used for both creative thinking and ordinary thinking (102). These cognitive components are:

- remembering
- imagining
- planning
- anticipating
- judging
- deciding
- determining
- perceiving
- comprehending
- recognising
- interpreting (Weisberg 106).

Since ordinary cognitive components and creative cognitive components can not be distinguished from each other, it is not possible to highlight any specific activity which needs to be supported by a process or tool above any of the other activities. All of these activities form part of ordinary thinking as well as creative thinking. Guilford's list of abilities explore specific abilities which creative people exhibit. In his view recognising these abilities will allow the identification of children who have creative potential. These abilities are:

- sensitivity to problems
- analysing ability
- ideational ability - the ability to form ideas
- ideational fluency - the ability to produce many ideas
- reorganizing or redefining ability
- flexibility of set
- span of ideational structure - how complex or intricate the ideas are
- ideational novelty - the ability to produce novel ideas
- evaluating ability
- synthesizing ability (Guilford 444-454)

To make sense of a long list of activities or abilities, creative thinking is sometimes categorized into two categories: convergent thinking and divergent thinking. Briefly convergent thinking results in a single solution, the type of thinking that chooses the best possible solution and is capable of judging which is the best solution. Divergent thinking results in many possible solutions to a problem. Divergent thinking is the type of thinking applied when a solution that diverges from what has been done before is required (Weisberg 96). The models by Finke,

Ward and Smith and Runco and Chand incorporate both types of thinking in different parts of the cycle. In the generative/ideation stage of the model multiple solutions are generated. In the exploratory/evaluative stage of the model solutions are assessed and discarded as needed. Examples of cognitive processes associated with the generative part of Finke, Ward and Smith's model (Finke, Ward, and Smith) are:

- retrieval
- association
- synthesis
- transformation
- analogical transfer
- categorical reduction

Examples of the cognitive processes associated with the exploratory/evaluative part of Finke, Ward and Smith's model (20) are:

- attribute finding
- conceptual interpretation
- functional inference
- contextual shifting
- hypothesis testing
- searching for limitations

According to some researchers even the separation of divergent and convergent thinking is regarded as an oversimplification and the detail of the cognitive activities are more complicated (Dietrich 23). The skills and abilities in the pre-

ceding paragraphs can be used in a variety of creative situations. Many if not all are applicable to projects where artwork is created using software.

Cognitive activities that form part of the creative process do not differ from ordinary cognitive activities so specific activities cannot be isolated since all the activities need to be supported by a the software artwork development process. Guilford's abilities are useful to recognise, but again, for the purpose of software artwork creation these abilities should be supported.

Even if the separation of creative thinking into convergent and divergent thinking is regarded as an oversimplification, it is useful in conjunction with documented models to ensure that a proposed software development process can accommodate a suitable range of cognitive activities. In practise the different activities overlap and the different phases may not be as clearly distinguishable so it may provide an incomplete description of the creative process. However as a tool to investigate the suitability of a development process, the iterative models and the convergent/divergent categorisations have merit as the goal is not to capture the creative process but rather to highlight specific aspects which must be supported by a development process.

2.4 Research on art and technology collaboration

The discussions thus far have not referred to any specific creative project and the creative process is complex and varied. So the practical investigation of cognitive skills, abilities and approaches in the context of software artwork creation is needed as concrete examples. Research by Candy and Edmonds, which is informed by Finke, Ward and Smith's model, has been done on the creative pro-

cesses of software artwork projects (134). This research is presented in the article *Modeling co-creativity in art and technology* (2002).

Candy and Edmonds investigate the different cognitive styles and how they interact in a practical situation where digital artworks were created in collaborative projects (139). Cognitive style is separated into five categories with binary descriptors for each category typifying the extremes of cognitive style in the category. The descriptors were derived from project observations and interviews with team members. Table 1 shows each style with its descriptors.

Cognitive style	Descriptor 1	Descriptor 2
Approach	Exploratory	Goal Driven
Role	Different	Same
Ethic	Art-led	Technology-led
Control	Important	Necessary
Methods	Traditional	Digital

Table 1: Summary of cognitive styles used by Candy and Edmonds

Source:LInda Candy and Ernest Edmonds Modeling co-creativity in art and technology (Proceedings of the 4th conference on Creativity and Cognition:2002) 136 Web

Approach means the approach that the team take to the project; whether clear goals are set at the beginning with little deviation or whether the ideas are explored and generated and a solution found as part of a process. Art-led ethic means that the best solution from an audience awareness and personal engagement was pursued. Technology-led ethic means that the optimal technology based solution was chosen (Candy and Edmonds 136).

Candy and Edmonds investigated the success of different projects and then connected this to the different cognitive styles which they used to analyse the projects (138). The approach, as cognitive style, that a project takes is a possible aspect which differentiates software art development processes from non-art

software development processes. The approach of the project is categorised by two descriptors, exploratory and goal driven. The exploratory approach is one where a "rough idea" of the end product is decided on in the beginning but the project evolves by exploring different avenues iteratively. The goal driven approach is one where clear systematic steps were followed with clear goals determined at the beginning of the project. From a creative cognition point of view using an exploratory approach does not preclude using a goal oriented approach and that both approaches can be useful depending on the situation (Candy and Edmonds 140).

The experimental approach used by artists is also noted by Trifonova, Jaccheri, and Bergaust in an extensive study of practitioners reports of 25 interactive artworks in the article *Software engineering issues in interactive installation art* (2008). This study was done to investigate the software engineering aspects of these projects but "experimentation as a style of artists" was identified as a characteristic which is peculiar to artistic projects (60). From the research on practical software art collaborations, the cognitive approach is taken as an element which has to be accommodated in a software development process. This means the software development process should be able to function with an approach that ranges between the binary opposites of exploratory to goal driven.

2.5 Characteristics of tools and process which assists creativity

Artists use computers to create software art. In addition a software development process can also be seen as a tool to support development. Recommendations for tools that assist creativity provide guidelines which can also be applied

to software development processes. Buss analysed the creative process rather than looking at creative artefacts or the creative human in his dissertation *Behavioural Patterns for the Analysis of Creative Behaviour* (2011). He used computer tools to track and map of the creative process. One of his goals was to identify concepts which support the creative process for any domain in which it happens (Buss v). He quotes Sawyer and notes properties about everyday creativity as observed when studying musical and theatre performance. Six elements stand out:

- collaboration,
- improvisation,
- can not be planned or revised ahead of time,
- emerges unpredictably from a group of people,
- depends on shared cultural knowledge,
- the process is the product (Buss 16).

To connect to this Candy and Edmonds have recommendations for successful collaborative projects. They stress strong communication skills. They also suggest that solutions which suit the technical aspects of a project but do not match the artistic vision of the project should be avoided. In addition they have the following recommendations:

- that a shared language be developed among team members
- that common artistic goals be developed
- that discussion of alternative solutions are encouraged
- that time is allocate to build team relationships and to "recover from mistakes" (141)

In a further study Edmonds et al. discuss recommendations for tools, specifically computer assistance, which are used in projects. The salient point being that the tools have "flexible interfaces which do not disrupt flow of thinking and action" (456). Since the software development process involves working closely with computers as tools, recommendations for computer assistance is relevant if the goal is to avoid any barriers to creativity. Creativity can be supported by providing tools that has "the capacity for users to rapidly generate multiple alternatives, explore their implications, or revert to earlier stages when needed" (Shneiderman 2).

Buss further suggests guidelines for tools that support creativity (32):

- support exploration
- support experimentation with many alternatives
- low threshold, that is be easy to learn
- support variability, that is there should be many ways to achieve the same thing
- support collaboration
- support integration of different tools
- be simple
- be careful about which areas of the tool is open to exploration and which is close
- be something that the developer would also want to use

Since collaboration is a large part of successful creative endeavours, any barrier which could jeopardise collaboration should be avoided (Fischer). Fischer defines the following barriers:

- spatial - collaborators are separated by space
- temporal - collaborators are separated by time zones
- conceptual - collaborators do not share common understanding
- technological - collaborators do not share similar domain orientated tools or software systems.

The spatial and temporal barriers can be alleviated to an extent by mediated communication technologies. Examples of such mediated and internet enabled technologies are: email, voice, text and video communication software (Fischer).

From these recommendations a development process or tool must not be a barrier to collaboration and must improve communication in the team. Furthermore the development process must be flexible, must support experimentation and must be simple to use. In addition a development process tool must allow for emergent creative idea development. These ideas can be used to assess the suitability of a proposed development process.

2.6 Conclusion

Various definitions of creativity overlap with the concept that something novel has to be created for the process to be creative. The models used to describe creativity have evolved from a four stage model to models which iterate and accommodate convergent and divergent aspects of the process. A list of everyday cognitive skills and abilities are used in the creative process. When looking in particular at software art projects, the cognitive approach can be categorized as goal oriented or exploratory/experimental. Recommendations for tools and processes that support creativity suggest: support for collaboration and communication, support for easy generation of multiple solutions and experimentation, flexibility for different ways of working, be simple.

3 Software Development Process

The process of software development for commercial purposes has been observed and documented. Early models of software development process focused on the different phases of the development and tried to make the process work no matter who took part in the process. The development cycles were long and did not adapt to change (Beck 70). The main goal of these processes was to be predictable so that the team delivered what was planned at the outset (Abrahamsson et al. 12; Novikov and Heuser 3). Highsmith and Cockburn explain that, what they call, "traditional processes" focus on obtaining clear requirements as far as possible on the outset of the project so that little variation occurs and an attempt is made to eliminate change during the project (120). Beck views the Waterfall process as a process where requirements needed to be clarified upfront (70). (See fig. 4.)

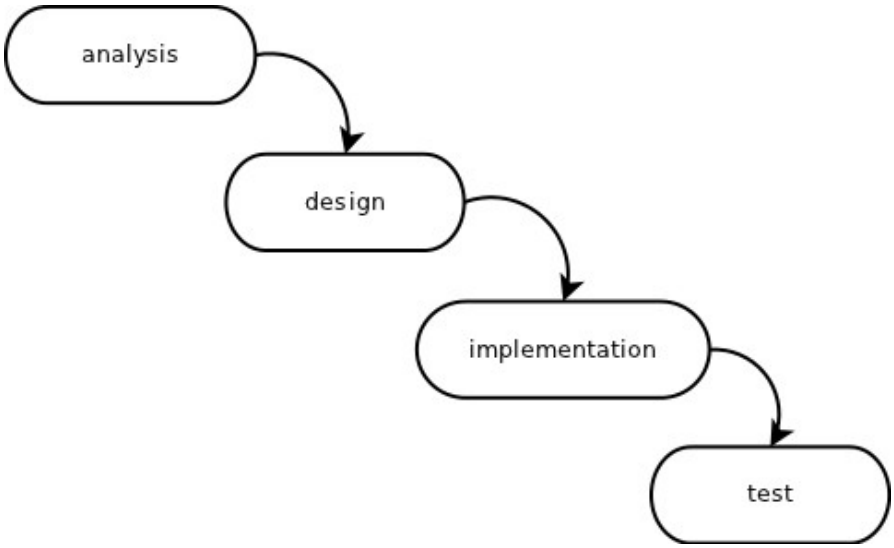


Fig. 4: Simplified Waterfall development model

Further investigation of the process uncovered that one of the main problems was that the requirements would change during the process. This change was inevitable and no amount of process put in place could avoid this; the changes that occur could be outside of the control of the project members. To accommodate the change, the process had to be adaptive. If the model wasn't sufficiently geared to accommodate changes in requirements, it would result in the team producing a product that worked as planned but that was no longer competitive (Highsmith and Cockburn 120). The longer a process development cycle took the bigger the chance that the requirements will change. A family of software development methodologies that was developed particularly to deal with changing and vague requirements is the Agile software development process (Abrahamsson et al. 14).

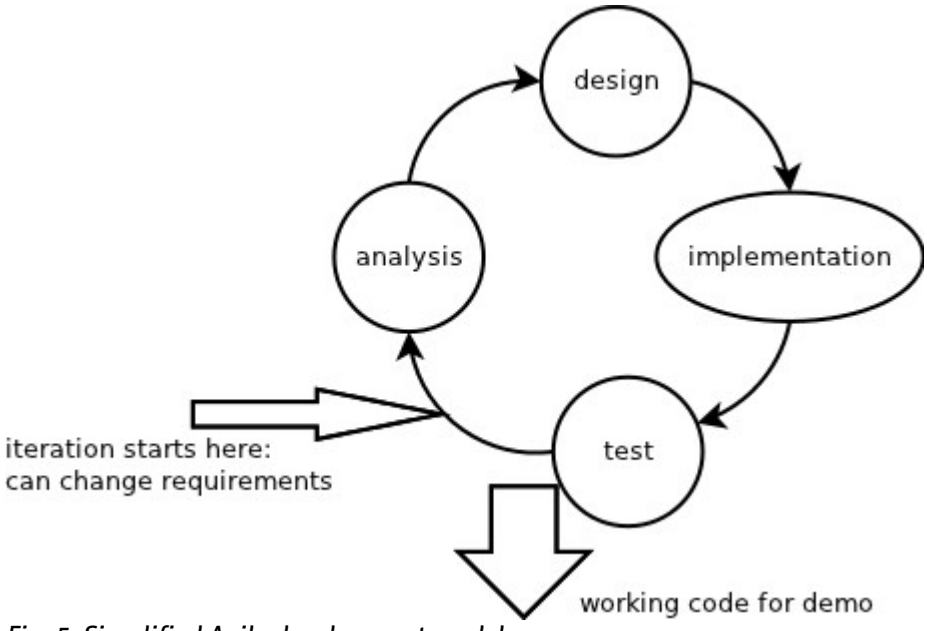


Fig. 5: Simplified Agile development model

One of the core elements of the Agile model is iteration and quick response to changing requirements so since digital art projects are characterised by vague requirements and an iterative approach due to the creative process, the Agile mod-

el is intuitively a better fit for digital art projects. (See fig. 5.) For this reason the Agile process will be investigated rather than traditional software development processes. Both Agile and traditional software development processes can be used to model the development process. Both provide tools and guides to support team communication and management as well as recommendations for testing and configuration management of the source code. The usefulness of the Agile process for use in digital art projects have however been confirmed by practitioners because it provides tools to manage project scheduling and project monitoring (Marchese; Bunt, "Risking code" 37).

The Agile values and principles are based on the *Manifesto for Agile Software Development* which was written by a group of software developers (2001) (Beck et al.). One of the developers who contributed to the *Manifesto for Agile Software Development* is Kent Beck. Kent Beck provides information about the XP methodology in the paper *Embracing change with extreme programming* (1999).

This chapter will look at the Agile family of methodologies, rather than the traditional development process, with particular focus on areas which will be useful for a digital art project. It will take the Agile values on which all Agile methodologies are based and discuss if they are applicable to digital art projects. It will then look at the Agile principles and identify a subset which are more appropriate for digital art projects. The roles of stakeholders specific to digital art projects and how they compare to software development project stakeholders will be identified to provide context in the discussion. Agile practices which can be useful for digital art projects will be discussed. Three practices, which stem from the Agile values and principles and are applicable to digital art projects, were chosen: iter-

ative process, regular feedback and continuous testing. Lastly software development practices which are not part of Agile but which can be useful in the digital art project context will be discussed.

3.1 Different Methodologies

Based on Agile values and principles more than one process methodology or philosophy have evolved. Shore and Warden describe the situation as follows: "Agility is an umbrella term for a variety of methods, most of which came about long before the term 'agile' was coined" (59). Examples of methodologies are: Scrum, Extreme Programming (XP), Crystal, Adaptive Software Development (ASD) and Feature Driven Development (FDD). Some of the documented methods propose a philosophy and others offer pragmatic suggestions of suitable activities and processes. Each of these process methodologies have their focus area and strong points. A useful comparison of the current state of the software development can be found in Abrahamsson et al. This is a comparative work which has the goal of reviewing and comparing the current documentation of Agile methodologies and is titled *Agile software development methods – Review and analysis* (2002). All of the methodologies discussed by Abrahamsson et al. share the following characteristics:

1. delivering something useful,
2. reliance on people,
3. encouraging collaboration,
4. promoting technical excellence,
5. doing the simplest thing possible and

6. being continuously adaptable (93).

The Scrum methodology provides processes for project management but does not provide much information on acceptance testing. The aim of ASD is to "provide a framework with enough guidance to prevent projects from falling into chaos, but not too much, which could suppress emergence and creativity." ASD also proposes that high-risk areas should be done first ASD does not offer advice on team roles and practical tasks which comprise the process. XP is the most documented methodology. It places emphasis on development practices and has less information on project management practices (Abrahamsson et al. 71-93).

The ASD methodology, has been useful for a software art project (Marchese). Since projects differ widely, all tools may not be suitable (Riehle 2). Abrahamsson proposes a combination of methodologies so that the strengths of different methodologies can be combined (75). Rather than focus on a specific method, a list of common Agile process element have been chosen and the recommendations from different methods and the relevance to software art projects are discussed for each element.

3.2 Agile values applied to software art

The Agile development process is based on core values and principles which were documented by a group of software professionals in the industry. The values are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation

- Customer collaboration over contract negotiation
- Responding to change over following a plan (Beck et al.)

These values are not phrased as absolutes but rather as guidelines, for instance the principle of communication before documentation does not mean there is no documentation just that the communication takes a higher priority (Novikov and Heuser 4). Each of these values can be viewed with regards to a digital art project. To value individuals and interactions will allow adjustments for the different ways that artists work without enforcing a specific development process. To value working software over comprehensive documentation is useful if the artwork and not the development process is the focus. It will also provide quick prototype feedback to artists which will support the creative process. Team collaboration and customer collaboration is especially important because digital art project teams are often made up of people who come from technical and art fields. In addition collaboration with communities who create software tool-kits for re-use also benefit from collaboration. The specific roles of stakeholders and the definition of the customer in this context will be discussed in detail later in this chapter. Being responsive to change will ensure that the process maintains flexibility during experimentation phases. Experimentation support and flexibility is important to support creative processes.

3.3 Agile principles applied to software art

From the Agile values, the Agile Manifesto lists a set of principles (Beck et al.). I highlight those principles which are applicable to this research:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- **Welcome changing requirements, even late in development. Agile processes harness change or the customer's competitive advantage.**
- **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.**
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals.
- Give them the environment and support they need, and trust them to get the job done.
- **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**
- **Working software is the primary measure of progress.**
- Agile processes promote sustainable development.
- The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- **The best architectures, requirements, and designs emerge from self-organizing teams.**

- **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.**

The Agile process is characterised by short iterative cycles where the team prioritises and establishes and possibly changes what needs to be done in each cycle. The deliveries are incremental. The teams are self-organising so the team determines how work is best done. The processes and work-structures are determined during the project as opposed to pre-determined, which makes the method emergent (Boehm and Turner 16). An experimental way of developing artworks can be accommodated within this system as the iterative nature of the process allows for adjustments from one cycle to the next. Artist-technical collaboration and solo artist work may benefit from being self organised and if these teams grow organically and not from an externally imposed structure, the teams are self-organising. Artwork projects are often combinations of techniques and ideas which have not been implemented before, so the characteristic of the Agile process that processes and work-structures are not pre-determined is applicable to art software projects.

Abrahamsson et al. have formulated a definition for Agile methods:

What makes a development method an agile one? This is the case when software development is incremental (small software releases, with rapid cycles), cooperative (customer and developers working constantly together with close communication), straightforward (the method itself is easy to learn and to modify, well documented), and adaptive (able to make last-minute changes) (19).

Abrahamsson et al. quote Cockburn explaining that Agile projects should have "light but sufficient rules of project behaviour" (15). Not all projects are suited to

the Agile process. Cockburn provides indicators or characteristics of software projects which increase the chance of potential success.

- Two to eight people in one room - Communication and community
- On-site usage experts - Short and continuous feedback cycles
- Short increments - One to three months, allows quick testing and repairing
- Fully automated regression tests - Unit functional tests stabilize code and allow continuous improvement
- Experienced developers - Experience speeds up the development time from 2 to 10 times compared to slower team members (Abrahamsson et al.15).

Some of these indicators can be applied to digital art projects although not all can be guaranteed. One indicator that can be proposed is the short increments as this fits well with the creative process. This indicator will be discussed in detail in subsequent section Iterative process on page 41. One indicator which will typically not be achievable is fully automated regression testing. A benefit of automated regression is that the tests are run regularly and not in a single project phase late in the development cycle. Due to the nature of digital art projects it may not be feasible or appropriate to build automated regression test. Testing will be addressed in the section Continuous testing on page 43. The other indicators, small co-located team, on-site experts and experienced developers can be seen as potential risks or benefits on a project by project basis.

3.4 Stakeholder roles in digital art projects

It is useful to identify the stakeholder roles for digital art projects, because when a topic is discussed it refers to the stakeholders and the stakeholders in a digital art project are not the same as the stakeholders in a software development project for commercial purposes. Trifonova, Jaccheri, and Bergaust discusses the roles that the different team members of a digital art project take and how this relates to the roles in a development team. See fig. 6. Although the stakeholders and their roles are mentioned as separate people, the roles may overlap in practise (Trifonova, Jaccheri, and Bergaust 9). The roles that are identified are: the artist - responsible for system requirements and content, the hardware and software developers - responsible for the construction and implementation of the artwork and the audience - responsible for interacting, providing input and experiencing output of the artwork. The artist can also be seen as the client because the requirements, the quality attributes and the schedule is driven by the artist. In addition sponsors or commissioners may also influence budget and schedule decisions (Trifonova, Jaccheri and Bergaust 51).

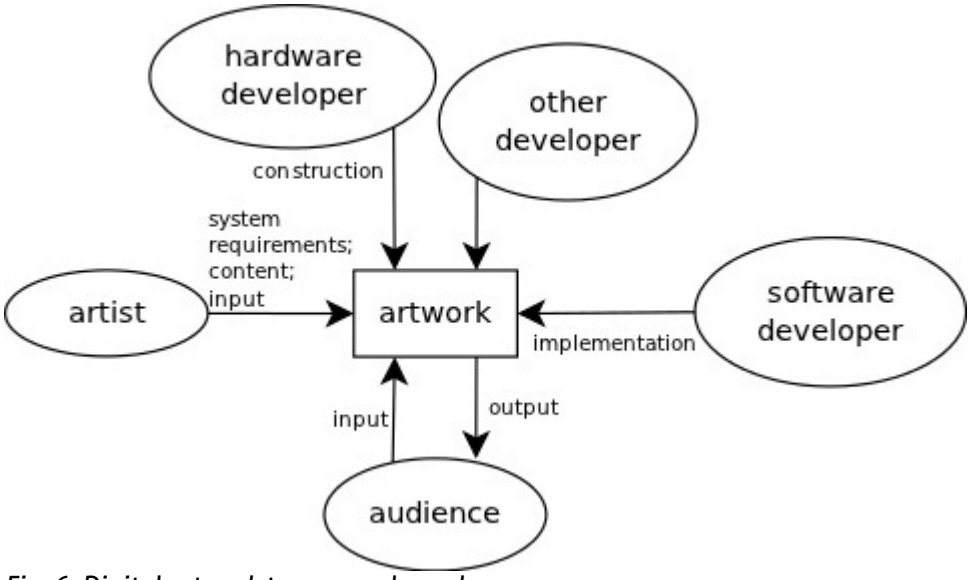


Fig. 6: Digital artwork team member roles

Source: A. Trifonova,L. Jaccheri, and K. Bergaust Software engineering issues ini interactive installation art (International Journal of Arts and Technology:2008) 51 Web

The stakeholders from a software development point of view and the roles from a digital art project point of view are compared in Table 2.

	artist	technical developer	audience	sponsor/ curator/ commissioner
customer	is			can be
developer	can be	is		
team manager	can be	can be		
tester	is	is	can be	can be
end-user	is		is	can be

Table 2: Digital artwork team stakeholder roles

The customer is someone who provides the requirements, sets the schedules and deadlines and provides resources and funding. This explains why the artist is the customer and if there is a sponsor or commissioner they could be a customer

as well. People who commission artwork usually have less say about the specific visual and conceptual requirements of the work as this is the role of the artist. In commercial software development, the end-user is the person who uses the software that was developed as a tool. In a digital art project the end-user is the audience but the software developed is not necessarily a tool. The artist can also be seen as an end-user as it is the artist's ideas and concepts which are communicated with the project. The blurring of the roles can imply that the artist is also the technical developer and is also the commissioning agent or curator. In addition the customer could be the developer, team manager and tester and end-user.

Looking at the interviews with artists for this research and the artwork documentation, the team stakeholder roles can be identified.

Stern collaborates with artists from different disciplines in works such as *Given Time* (Stern, "Nathaniel Stern : Given Time, Networked Installation and Continuous Performance, February 2010"). In Stern's work *stuttering (Stern, "Nathaniel Stern : Stuttering, Interactive Installation, 2003")*, the artist is the technical developer and the tester. Since this is an interactive installation the audience is the end-user and the tester.

Goldbergs custom software tool, *Dervish* is an example where the artist is the customer, the developer, the tester and the end-user (Goldberg, "Dervish"). Some of the projects of the artists that were interviewed were commissioned so for some projects the customer was the artist and the commissioner. An example of a project like this is *Abstract Microecologies* by Pierre Proske as this work was done as part of an artist residency at the Department of Archaeology and Natural History at the Australian National University (Proske, "Abstract Microecologies").

Thayer and Bunt did not indicate that they collaborated. They took the role of customer, technical developer, sponsor and tester. Both Thayer and Bunt are aware that the end-user experiences the work in a different way. Bunt states that his code is indirectly visible as a visual artefact. Thayer makes his code very visible and his work is not accessible without accessing the code. In his case he says that the medium in which the end-user experiences his work is different to the medium that he uses to create his work.

In the case studies there are examples of artist performing the roles as described in Fig. 6 and Table 2. There are examples of the artist as customer, as technical developer, as team-manager, as tester and as end-user.

3.5 Agile process phases applied to software art

The process phases of Agile can be seen in terms of process steps which artists may use. A diagram of the Agile process steps adapted to software art can be seen in Fig. 7.

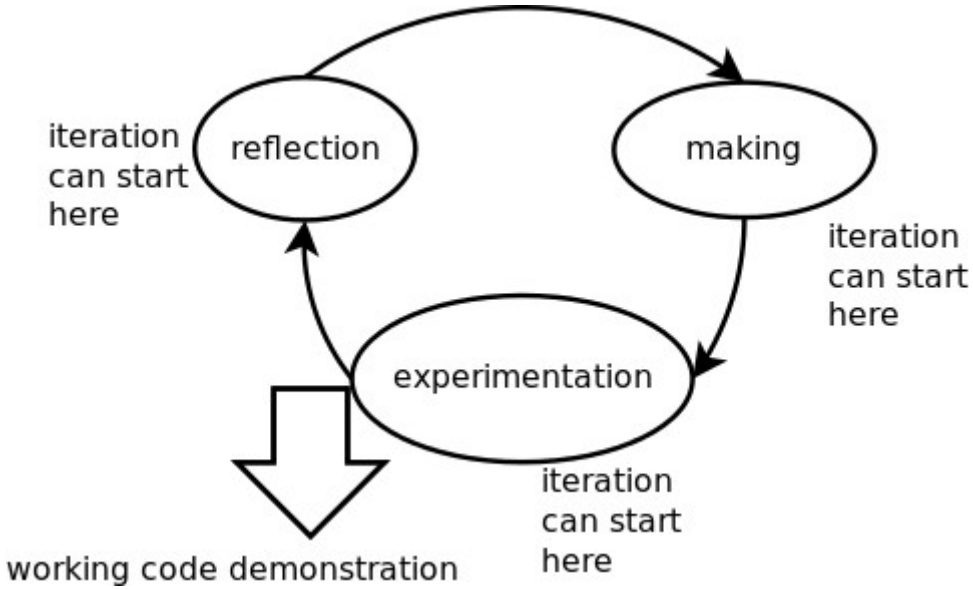


Fig. 7: Agile process phases applied to software art

The analysis and design phase of the agile process is the process step which involves reflection on the goals of the project. It requires prioritisation and contemplation of the features or elements which will be included or excluded. In this phase the goals of the project and the next iteration are questioned and compared to artists or commissioners requirements. For an artist, taking the role of customer in this phase, this step may include a reflection and analysis of the conceptual underpinnings and goals of the artwork. This phase re-visits the constraints of the project and influences the subsequent phases. This phase is informed by the preceding implementation and testing phases.

The implementation phase of the process is the step where the code is written. It is the making part of the process for the artist. Testing and demonstration of the implementation can happen concurrently or after the making phase. The testing aspects of the Agile process and demonstrating working code is the part of the process where the artist engages with what was made. This engagement may involve experimentation and exploration. The making(implementation) and engagement(testing) activities then inform the reflection(planning) of the next iteration. In the preceding chapter, these steps are teased apart as separate steps to aid understanding and description but they may occur continuously and overlap in practice. What makes Agile different to preceding development processes is that it proposes shortening the time between reflection, implementation and engagement and having multiple reflection point during the project. Because of the feedback mechanism the process is adaptable. From an engineering viewpoint, the potential problem with making short cycles is that the time overhead for reflection becomes larger and the making and engagement time becomes less.

From an artist perspective though, if the strategy of the artist is to explore the concepts and ideas of the project, the time spent in reflection and planning is an integral part of the project process.

From an artists perspective the Agile process involves reflecting and making choices, making and engaging critically with what has been made. The process needs to stay adaptable.

3.6 Agile development practices relevant to digital art production

3.6.1 Iterative process

Agile value: *Working software over comprehensive documentation.*

Agile principle: *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.*

Using an iterative approach is core to agile development. It stems from both an Agile value and an Agile principle. Using iterations also allows changing requirements to be managed. Requirements are chosen at the start of an iteration period and not changed until the iteration period is completed. This allows the development to experience short bursts of fixed requirements but also gives a regular opportunity for changing requirements to be accommodated. The beginning of an iteration is the point where the customer can add or change requirements (Shore and Warden 43; Highsmith and Cockburn 121). Since the Agile value focuses on working software it means that it is ideal if a working demonstrable version of software is created at the end of each iteration. It also means that the project is

tested as far as possible at the end of each iteration. The length of an iteration is variable. The team should decide the length of the iteration and different methodologies advocate different iteration lengths. In the case of Scrum and XP the iteration length stays the same from iteration to iteration. In the case of Feature Driven Development, the iteration lasts until the feature is complete. This could be 2 days or 2 weeks (Abrahamsson et al. 52). Shore and Warden explain that activities which would happen in different phases in older methodologies happen in each iteration in an Agile approach. That means requirements, design, development and testing happen simultaneously during each iteration. The benefit of this is that feedback is quicker because the team does not have to wait until late in the project life-cycle before test are done (18).

A short iteration cycle is a natural action in software that has a strong visual component such as coded artworks. The overlaps and similarities with the creative process and how it relates to the peculiarities of coded artwork will be discussed in chapter 4.

3.6.2 Regular Feedback with stakeholders

Agile value: *Individuals and interactions over processes and tools.*

Agile principle: *Business people and developers must work together daily throughout the project.*

Agile process propose regular brief meetings with stakeholders during and iteration. This means regular involvement and meetings among the artists and developers if their roles are not overlapping. These meetings are not used to change requirements but to strengthen communication during the iteration. Regular

feedback at different intervals allows the process to adjust quickly to changes in requirements or environment. Agile practices build the response to change into the way the team works; the practises embrace change, rather than avoiding or resisting change (Highsmith and Cockburn 122). Feedback is the mechanism used to detect and to respond to change.

Agile process also require end of iteration feedback. Again this may take the form of a demonstration and also planning for the next iteration. Some process like Scrum allow all stakeholders to specify actions which should be taken or tasks for the team (Abrahamsson et al. 34). Since the test audience and the commissioner or curator can also be seen as a stakeholder, these meetings would be suitable to get feedback from test audience members at an early stage of the development cycle.

In practice, teams that develop software art projects may come from multiple disciplines. Regular feedback sessions support communication and collaboration among people with different backgrounds and can create a clearer understanding of project goals and progress. Regular feedback sessions can also be used to explore the system being developed so that the requirements that need to change can be discovered. For the situation where the artist is the customer, the developer and the tester, the regular feedback may be less explicit and take the form of reflection.

3.6.3 Continuous testing

Agile value: *Working software over comprehensive documentation.*

Agile principle: *Working software is the primary measure of progress.*

Working in an Agile way requires continuous or constant testing as working this way will allow the team to detect defects earlier (Abrahamsson et al. 17). Testing also ensures working software which is one of the Agile values. Testing is a development practice that is not specifically Agile. Testing will be discussed in the section relating to general development practices. Agile methodology does propose continuous testing during each iteration and automated tests. Shore and Warden proposes automated tests if possible for all tests except acceptance tests (25,288).

The Scrum process proposes tests at the end of each iteration and system test when the requirements are completed. The tests done at the end of each iteration is a demonstration to the customer and will be called iteration tests in this section. These tests are not acceptance tests as the goal is to demonstrate what has been completed in each iteration and to provide a feedback opportunity. Likewise the XP process has testing as a key part of the iterations towards a release. XP also proposed additional system performance tests in the production phase of development. It is the responsibility of both the programmer and the customer, or artist in our case, to define tests. XP proposes a test driven approach, where the tests, typically unit tests, are written before the code is written.

This way of working is appropriate where automated tests are written and maintained. XP and Clear methodologies propose automated tests as essential to project success. Software art projects may not implement automated tests as one of the benefits of automated tests are that they prevent regression problems. Regression problems occur when software is changed after features have been completed and a seemingly unrelated change cause a functional area to break. If the

software art project requires a once off installation but is not used by multiple customers over a time with version updates, automated regression tests would not give any benefit. Furthermore digital art projects can involve a large visual and audio component, the output of the work can be ambiguous as part of the artist's strategy and the output can involve random elements. These factors make automated tests expensive and difficult to implement and may not give enough benefit to justify the implementation.

Even if automated tests are not implemented, testing can be seen as a form of demonstration or prototype to the customer (artist in our case), early and regular testing can assist communication in the team. Regular demonstration with test audience members at the end of each iteration can assist in ironing out interaction problems which may be easy to fix early on in the project but be more difficult to fix at a later stage. In addition if the definition of what the system should do is described in the form of tests, the creation of tests before implementation may assist the customer/artist to uncover the requirements of the work. Regular testing opportunities can also allow the customer/artist to play with the system to explore the limitations and idiosyncrasies of what has been created. Recording the result of testing activities provides useful information for process and software adjustment and can form part of creative research.

3.6.4 Collaboration

Agile value: *Individuals and interactions over processes and tools.*

Agile value: *Customer collaboration over contract negotiation.*

Since collaboration is a core value of the Agile system, all methodologies value this aspect. Scrum encourages collaboration through daily and end of iteration meetings. ASD divides the iterative process into three phases, speculate, collaborate and learn. The name of the collaborate phase is chosen to emphasise team work. The collaborate phase is the phase where "concurrent component engineering" occurs. This is the phase where the code is written and the systems are built. Shore and Warden's practical information on how to use XP describe the following activities to support collaboration: the team sits together in the same space, pair programming where 2 programmers work on a coding task at the same time consulting with each other and daily short meetings called stand-up meetings (101). Beck proposes XP for small teams (157). Collaboration, communication and team management become difficult in large teams. ASD views the organisation as adaptive systems where the practices attempt at creating an emergent order out of connected individuals (Abrahamsson et al. 91). The interconnection of individuals cannot happen if there is no collaboration. In Scrum practices the teams are also self organising and independent. This allows each team to adjust to the requirements and the environment. Digital art projects that are developed in a team require good collaboration and communication skills especially if the team contains members who come from different backgrounds.

3.7 Additional development practices relevant to digital art production

3.7.1 Configuration Management

Configuration management is the systems used to capture software changes and to keep track of the history of changes on each source file (Abrahamsson et

al. 56). From an engineering point of view configuration management is used to ensure that software can be created in a repeatable way. It is used to track all changes and allows the programmers to revert to a known working point in the software. Configuration management ensures that no work is lost. If the commenting system of source code configuration management systems are used, it is also a way of recording why things have changed. Shore and Warden describes the benefits of configuration management and software version control as allowing concurrent editing of files so more than one team member can change files without waiting for another, time travel or the ability to reconstruct a version that worked before and the security of backing up the source code as well as the history of the changes (169).

This element of software development is not addressed explicitly by the Agile values and principles, however it is advocated as indispensable by more than one process, especially the methodologies which focus on day-to-day development practices such as XP and FDD. The Scrum process proposes daily software builds against the latest version under configuration control.

To track each change made to the software may seem like unnecessary overhead for a project that is attempting to follow the minimum amount of formal process to remain Agile. If the source code change tracking is seen as a tool that allows the programmer to retrieve any earlier experiment without destroying what has been created to date, it becomes a tool which supports a process which needs to change quickly to changing requirements. From a software art project point of view, recording a history of previous versions allows the developers to

explore multiple solutions without jeopardising what has been created already. This allows for non-destructive experimentation.

3.7.2 Testing

Testing is a software development practice where the operation of the software project is investigated. Testing activities that focus on the verification aspects of software testing use the following definition: "Testing is the process of executing a program with the intent of finding errors" (Myers et al. 11). There are other definitions of testing. According to James Bach "Testing is questioning a product in order to evaluate it" (Bach, "What is exploratory testing?"). According to Kaner "Testing is an Empirical technical investigation done to provide stakeholders, information about quality of a product or a service" (Kaner). The definitions of these two authors are quoted because they are advocates of a testing approach called exploratory testing. Exploratory testing is a type of testing where the tests are designed and executed at the same time. This is distinguished from scripted testing where testing depends on test procedures that are written ahead of time. Exploratory testing is free form and may be a good fit for the way that software art projects are developed. There are situations when exploratory testing is appropriate. Some of these situations are: when "rapid feedback and learning of the product is needed", when there is not enough time for a systematic approach, when the status of specific risks needs to be assessed, when diversity in testing is required and when testing from the end-user point of view is needed (Bach, "Exploratory Testing Explained"). Since the artist is often the tester and tests the project from the end-user perspective, this approach is appropriate. In

digital arts projects, testing the code can be used to assess risks. This kind of testing is also used to learn the capabilities and behaviour of the software. This is another reason why this testing approach is applicable. Often in digital art projects, deadlines are not flexible so there may not be time for systematic testing. If the testing of the software is seen as a creative activity it compliments the exploratory approach that digital art projects may have as a creative process approach. Testing in this way then becomes part of the creative practice. If tests done in this way are documented the documentation can form part of the artist's creative research process.

Exploratory testing has been criticised for not providing feedback on the progress of the coverage of the test and not providing feedback on the progress of an individual tester (Itkonen and Rautiainen 87). Since the artist takes the role of customer and the artist is often the tester, the progress feedback directly affects the customer acceptance of the project. Since digital art projects usually don't have rigorous accountability requirements for acceptance testing, these criticism may not affect digital art projects.

Whether scripted or exploratory, there are different kinds of tests which are done for different reasons. Some of these test types are: customer acceptance tests, unit tests, functional tests, integration test and end-to-end or system tests and acceptance tests. Unit tests are used to verify the correct operation of subsections of the code. Functional tests are used to verify that a particular function of the software works correctly. Integration tests are used when different parts of the solution are integrated, for example different software systems that need to talk to each other over a network. System- or end-to-end tests verify that the sys-

tem as a whole with all connected sub-systems work together (Shore and Warden 303). Customer acceptance tests are seen as a way to communicate the knowledge that the customer has about the way the system should work. They are seen as examples of how the software should work (Shore and Warden 282). They can also be used to make sure that the system meets the requirements and is accepted by the customer. These types of test are all important for digital art projects but may sometimes be performed as part of the development process and not as explicit documented repeatable steps. All these types of tests can be done using the exploratory approach.

3.8 Conclusion

Many aspects of the Agile process are suitable for digital art projects because they are specifically geared for projects that experience changing requirements. The role of the artist and the audience were identified in terms of stakeholders in the Agile process. The artist is the customer but also the tester and possibly the end-user. The audience is the end-user. From the Agile values and principles four areas of the process were identified which have benefit for digital art: iterative process, regular feedback with stakeholders, continuous testing and collaboration. Two areas of general software development practice, which may help digital art projects, were identified and discussed. They are configuration management and testing. Both these areas can form part of the creative process. This chapter forms a starting point, from the software engineering perspective for the discussions, and comparisons in chapter 5.

4 Software as Medium

4.1 Software

Software is what makes the hardware behave or misbehave. It is the codified intention of the programmer which controls the behaviour of the hardware. The code is written by the programmer, compiled or interpreted by other programs (compilers and interpreters), linked to pre-compiled libraries created by other programmers resulting in an executable that runs on a machine. Software is described by Mahoney as "Elusively intangible. In essence, it is the behaviour of the machine when running. It is what converts the architecture to action, and it is constructed with action in mind; the programmer aims to make something happen" (12). The running program takes inputs, which may be sensor inputs, stored values or data received from other sources. The program then acts on the inputs following the algorithms and logic as coded by the programmer. It produces data which may be visuals, sound or inputs to other programs. Depending on the intention of the programmer and the framing of the result, the code or the running result or the output of the program, can be considered an artwork. Simon states "For me, what's important is that a piece of software can be considered an artwork, and that writing software is as creative as it is technical, and the choices made for language, data structure, methods, etc., are significant creative choices" (Ippolito and Simon).

4.2 Medium

The word medium, when it is used in relation to art, usually means "material used in a specific artistic technique" or "a specific kind of artistic technique". In the case of software artworks it could mean that software was the technique or material used to create the artwork. This could then mean that the software was the tool used to create the art but also that the artwork was made out of software. Another meaning of medium, which is also relevant to this discussion is, "intervening substance through which something else is transmitted" and also "An agency by which something is accomplished, conveyed, or transferred". A fourth alternative is "surrounding environment in which something functions or thrives" ("medium"). So if the medium of an artwork is software, it means that the technique or material which is used, is software. Furthermore it can mean that the intentions of the artist/programmer is carried by the software or transferred by the software. Software could also be interpreted as the environment in which the concepts of the artist/programmer functions. Choosing software as medium has implications on multiple levels. The work can be made with software, or use software to communicate concepts and ideas, or the resulting artefact could be software or software can provide the environment in which the artist's ideas function. Any or all of these possibilities can act at the same time.

When an artist creates a work, the medium may be software, but when the viewer/user experiences the work the medium may have changed to be a screen or a projection. For instance code is written on one machine, transmitted over the Internet to another machine, let's say a mobile phone, and decoded on the browser of the phone to present it to the viewer on a mobile phone screen and

mobile phone speakers. ".. the artist rely on code which .. can be squeezed through a modem line" (Blais and Ippolito). As Thayer explains it: "I don't think that 'my medium' is the same as the 'viewer's medium'. My medium is the code. That's what I shape and manipulate to convey my 'message'. The viewer's medium can be something else. It could be the Internet or the computer or the screen, depending on how they regard the work. It could even be the code as long as I reveal it. But I'm not really in a position to dictate to the viewer what they may or may not refer to as 'the medium'. That's dependent on their own experience" (Thayer, "On artists"). The implications of this is that the medium is not static. It can change over the lifetime of the work. The medium that the artist experiences when creating the work is potentially completely different to the medium that the audience experiences. Also the artist can not control all elements of the medium. Aspects such as scale, colour and rendering speed are examples of elements that may change when the medium changes. Hayles says the work is written and read in "distributed cognitive environments" (Hayles, "Print Is Flat" 81). Cognitive in this context means that the work is experienced in a technology, such as a browser, which contains logic that can modify the visuals that are presented to the viewer. What is meant by distributed is that the work is transmitted and received in multiple places. Manovich refers to the different audience experience possibilities as an example of the variability principle of new media, which states that a digital artwork is not fixed and has potential to have multiple versions (56).

This movement from one medium to another is a particular characteristic of software because it is in a digital form and can be transmitted. This applies not

only to images and sound but also to coded behaviour. Whitelaw explains as follows: "If the basic materials of the work are digital - that is, abstract patterns that can travel through any number of different substrates - then how do we make them perceivable? Or, how do we choose a mapping, a way of making data available to perception? Manovich calls this the "built-in existential angst" of digital media" (Whitelaw and Prudence) .

Marshall McLuhan's oft quoted statement "The medium is the message" is appropriate in this context. The medium has the capability of "reshaping and restructuring patterns or social interdependences" (Fiore and McLuhan 8). The medium of a software artwork can change. This mutability of medium allows the work to be seen in a different way to the way it was created by the artist. In some cases it also allows people across the world to participate in the artwork even if the are not co-located. So the meaning of the software artwork is influenced by the peculiarities of the medium. The medium is seen as an extension of ourselves. Therefore it has "social and personal consequences" (McLuhan 19). In this case it has personal and social consequences on artists and audiences. Aspects of the medium that can have personal and social consequence are: the ability to connect to different people in different locations instantaneously, the loss of control over how the art is experienced and the transience of digital artefacts. The medium changes the "scale, pace or pattern" of the way artists and audiences interact with each other. These changes affect the social interaction because time and spaces factors are changing (McLuhan 21). According to McLuhan these effects are independent of the content of the work (McLuhan 23).

Each definition of medium and how it connects with software art will be explored sections to follow. To find out how software as material or technique is used, interviews with academics and artists will be used as starting point to show some characteristics of the medium. These characteristics will be compared to principles of new media as defined by Lev Manovich and the concepts unique to code based writing as documented by N. Katherine Hayles. The section *4.4 Software as transmission medium* explores the connection that software art has with conceptual art and with language. The section *4.5 Software as functioning environment* investigates the way artists use software as metaphor, as muse and as simulation environment.

4.3 Software as material or technique

4.3.1 Characteristics of software as medium

Artists use different technologies for different reasons. Simon feels that technology provides a way of exploring ideas and providing different perspectives but that this potential is greater if the artist is able to program the technologies (Ippolito and Simon). Being able to manipulate the software gives more control over the medium and allow greater possibility for exploration. The choice to use software as medium may be because it has capabilities that fit the artist's strategy. It may also be because the choice of software supports conceptual aspects of an artist's work. The **programmability** of the medium is the first characteristic that I identify. From my reading: opinions of academics in the field and examples of reasons that artists give for their choice of medium, are used to identify additional characteristics of software as medium.

Alexander Galloway is an academic in the field of media, culture and communication and he is also a programmer (Galloway). He says "the biggest cultural ramification I see is that software is an action medium. Software does stuff. This is entirely different from literature, film, or other previous media." He quotes Friederich Kittler:"code is the first type of language that does what it says" (Wands 168). This view highlights the **non-static** nature of the software art work. It also re-iterates the linguistic nature of software art creation. The second characteristic then is non-static nature.

Mark Tribe is an artist who uses media technology. He is also an academic in the art and media field and he is the founder of Rhizome, which is an organisation that supports artistic practices that engage technology (Tribe). He views "appropriation as an artistic strategy combined with open source model of sharing code allows for easy re-use in code medium art works" (Tribe and Jana). This is important from a team/community communication point of view as well as from a code re-use architectural point of view. The software artwork is not necessarily a unique once off piece but something that can be **mutated** and **re-used** by other artists, if the code is accessible and re-usable and if people can find out what is available. This view illustrates the importance of collaboration for this artist and it means software processes which support collaboration may assist this artist. The third characteristic is that software can be mutated and re-used.

Joshua Davis is an acclaimed artist who uses technology to create his work (Kirsner). Davis states "Using a generative process allows for experimentation that would be time-consuming and impossible in analogue media. There's always a surprising sense of discovery with this process, because I'm setting up an envir-

onment and allowing a scenario to live within it" (Malmberg). Davis' view highlights the exploratory nature of his approach. He allows the action nature of the software to produce new material. Davis' opinion confirms the changeability of the requirements of a software art project. It also highlights the need for flexibility in the development process which is required to allow the discoveries to happen. The fourth characteristic is software's capability to **generate** multiple solutions. The characteristic of **flexibility** relates to software being mutable.

Another artist who highlights flexibility is Pall Thayer. Thayer is an artist who uses code as medium. His work Microcodes is a series of small code based artworks which convey their conceptual mean in the title, the reading of the code and the running of the code on a computer. For Thayer it is important to use a coding medium that is flexible which will allow him to develop a "distinctive style". Getting to know a medium better will allow the medium to become more flexible."It becomes like putty in their[artists'] hands, which they can easily shape into whatever they choose." He feels that computer programmers may require strict structures from their computer languages but that using a language which allows one to solve a problem in more than one way is more suitable to his process (see).

Although this list is by no means exhaustive, from the selection of artists and academics opinions the following characteristics of software art emerges:

- programmable
- non-static, active
- algorithmic and generative, can do repetitions easily
- mutable, flexible

- re-usable, remixing is easy

Authors such as Lev Manovich and N. Katherine Hayles have analysed digital based art work and formulated principles and media specific characteristics. Each of the characteristics which I have accentuated, will be discussed in relation to Manovich and Hayles views. The book *The Language of New Media* by Lev Manovich (2001) and the article *Print Is Flat, Code Is Deep: The importance of Media-Specific Analysis* by N. Katherine Hayles (2004) provide insight on the characteristics of the medium.

4.3.2 Programmable

Programmable means the software can be changed to change the behaviour of the work. It controls the behaviour of the hardware. Kittler sees this as a feature of the hardware that is programmable and not of the software (Kittler). Hayles sees the hardware as an analogue layer which is controlled by the software. Humans cannot perceive the effects of the software in the digital format. It has to be converted into a an analogue form, for instance visuals, light our sound (Hayles, "Print Is Flat" 78). So the programmability of the hardware is controlled by the artists using software to change the analogue output which is perceived by the audience. Manovich uses the term digitization for the conversion from continuous presentation to numerical presentation. Programmability is a result of the numerical representation of the media, according to Manovich's first principle. Once the media has been digitized, it can be manipulated algorithmically, or it becomes programmable (Manovich, Language of New Media 27-28). By manipulating the software the artist has control of the hardware and the analogue medium presen-

ted to the audience. If an artist controls the medium with software, skills relating to software programming and management of the software development process, can improve the control the artist has in exploring the capabilities of the medium. Skills used to manipulate software layers of a work, then necessarily affect the making of the work and the creative process. It is this programmable control that allows the artist to define non static behaviour of a work.

4.3.3 Non-static, active

Hayles points out that even if imagery that is created by a digital work and presented to the viewer remains static, it is still dynamic in that the fixed nature of the imagery has to be sustained actively by the software and hardware (Hayles, "Print Is Flat" 74). So non-static means that what is presented to the audience is not static but can change depending on different factors. It can change because of the behaviour programmed by the artists. It can also change as a result of the audience interaction with the work which modifies what is presented to the audience or as a response to the environment. This means as with non-digital work, the audience interacts on a conceptual level creating meanings. Furthermore code based artworks also allow for interaction on a physical level, directly modifying what is being presented. Manovich views the symbolic and conceptual interaction that happens with the work as internal and private. The physical or external interaction with the work is still controlled by the software and so, by interacting with the work, Manovich sees the audience following the "mental trajectory" of the artist (Language of New Media 61). The algorithms that the artist constructs controls the non-static behaviour of the artwork.

4.3.4 Running algorithms

Algorithms captured in software are a way of constructing sequences of operation which can then be performed by the hardware and not by a human. This supports Manovich's third principle, automation (Language of New Media 32). Examples of automation are generated web pages or generated 3D objects in game worlds. Algorithms provide a way to describe repetitions. Up to a point the machine can execute one or 1024 repetitions with the same amount of ease. This allows the artist to describe repetitive tasks, creating generative works which would be tedious and slow to execute by hand. It allows the artist to define a set of rules and then allows the computer to produce multiple variations, generating possible artefacts or outputs. This is a strategy used by Joshua Davis to produce complex works (Kirsner).

Automation does not apply only to artefacts resulting from digital artworks but also to behaviour. This then allows the artist to implement, through his choices, an active and non-static artistic form. The behaviour of an algorithm is controlled by starting conditions and changing conditions while the algorithm is executing. This makes the mutable character of software accessible to the artist. The algorithm can be defined ahead of time but it can also mutate on the fly.

4.3.5 Mutable, flexible

Because the code and the data are saved and transmitted in a digital form, there is no discernible difference between the original and the copy of either the code or the data. This is a direct consequence of Manovich's first principle: numerical representation (2001 27). So this means if a copy is changed, the original

is not destroyed. This fact makes software mutable. A small change in the code can cause significant changes in the behaviour. Data input to running software can be changed while the software is running, causing on the fly changes (Hayles, "Print Is Flat" 76). Small adjustments by the user or a change in a data element can change what is presented to the audience completely (Hayles, "Print Is Flat" 81).

Modularity, Manovich's second principle, states that a digital artwork can consist of independent parts, which again consist of smaller independent parts (Language of New Media 31). These smaller parts can be recombined in different ways. The recombination can be dynamic (Hayles, "Print Is Flat" 81). This mutability has creative implications at the design and coding stage (Bunt, "Risking code" 31). It means the artist can create code in small independent modules which can be recombined. The artist can explore multiple solutions. It also has creative implications for the execution of the artwork as it can contribute to the non-static and interactive aspects of the work. The same behaviour can be performed on different data sets. Generated data sets can be input to modify behaviour. This describes Manovich's fourth principle, variability, which relies on numerical coding and modularity. Variability means that the digital artwork does not have a fixed version but that there is potential for any number of versions (Language of New Media 36). Hayles phrases this phenomenon as "fragmentation and recombination". Software which runs can separate and re-combine bits of digital data in an active way ("Print Is Flat" 77).

Since the code and the data are in digital form as well as in modular format, it can be transmitted, recombined and shared with other artists. The modules of

data or code can be re-used in the original format. They can also be recombined and re-mixed to form new works.

4.3.6 Re-use, re-mix

The digital form of code and data and the identicalness of original and copy has further implication for sharing and re-use of material. If there is no difference between the original and the copy, a piece of content or a piece of code can be shared among people. Since both content and code is mutable, new artworks can be created based on older works by the same or different artists. Furthermore solutions to technical problems such as viewer detection can be adjusted and improved by multiple people without having to develop the solution from scratch. On one level content and data can be shared and re-used. On the next level code such as a patch in MAX/MSP can be shared. On the third level, if the source code of the tools are open as well, then the community can help improve the tools. The artist is also able to tailor and extend the tools to explore concepts that have not been addressed before. An example of open source tools and community is the openframeworks framework and software.

Ease of re-mix and re-use of data and code has implications for the artist and the audience. Incremental solutions are possible because an artist can use an existing solution for a technical problem to build on. Easy re-use and re-mix support collaboration among artists and technical developers. From a software development point of view re-use of modules may need modular software architectural design to facilitate re-use, so it means the resulting artwork will have a different software structure if the artist re-uses other people's code and if the artist wants

to contribute modules of the developed work to the community. The concept of the original is also challenged in this way of working. If open source components are used, the licensing requirements of modules that are re-used may require that the resulting code is open source as well.

4.3.7 Software as tool

In some cases the programmer-artist uses software only as tool. The software is used to create an effect which would not be possible or would be tedious to create using analogue means. In Lovejoy's view: "Although the coherence of the artists conceptualization process is the most fundamental aspect of art-making, the influence of tools and of technological conditions transforms the production and dissemination of art" (31). Indicating that software as medium is primarily concerned with only production and dissemination, but Hayles expands on this by stating; "But the computer is not so much a machine as it is a mind amplification tool and different kind of expressive medium" (Hayles, My Mother Was a Computer 60). This indicates then that the computer code itself is the medium in which an artist can create a software artwork. Mohr sees the computer as providing the "experience of a physical and intellectual extension of myself" (Edmonds and Candy 94).

To take this one step further Pall Thayer verbalises as follows:

"I would not say that USING software as a medium has changed my creative process. Rather that CREATING software as a medium has changed my creative process" (Appendix VII: Email interview with Pall Thayer). In some cases artists write software to extend a tool because they are exploring concepts which can-

not be expressed by the software tools as is. In other cases such as Thayer, artist create software as part of their process and as the software itself is the artefact of the process. Goldberg states that "it's the responsibility of the digital artist to hate his tools" (Appendix IV: Transcription of Skype Interview with Joshua Goldberg). What he means by this is that the artist should be aware of the limitations and restrictions that the tools are placing on the creative process and by transcending these limitations, the boundaries are expanded and something novel is created. Proske says he has be part of the coding process to engage with the medium and that is how he creates work that is unique to him. Although Proske feels there should be a balance between making artwork and modifying or making tools he acknowledges that his current process involves both (Appendix V: Skype interview with Pierre Proske).

Creating software and extending the tools then becomes an intellectual extension of the artist. Software is no longer a medium, in the technique or material sense of the word, but becomes an intervening substance that the artist uses to communicate ideas and concepts.

4.4 Software as transmission medium

4.4.1 Software art relates to conceptual art

The conceptual artist Sol LeWitt is quoted when referring to software art because he created the works Wall Drawings. These works consisted of instructions on how the drawings had to be performed. If someone bought a Wall Drawing it would be executed/performed in their house and re-drawn when they moved ("Oral history interview with Sol LeWitt"). The parallels with software are then

that the instructions are like the software and the assistants who perform/draw the works are like the hardware that runs/performs the code. For LeWitt it was more than just capturing the process in a series of instructions or the visual aspects of the final drawing. In his view the idea or concept is the source: "The idea becomes a machine that makes the art" (LeWitt, "Paragraphs on conceptual art").

Similarly for artists who use software the work becomes "the expression of an idea that becomes reality by simulating it" (Shanken 434). In the case of software art, especially interactive art, the viewer may take an active role in expression of the idea. So it is a combination of the concept of the artist, the software, the hardware and the viewer which, in Shanken's view, requires the viewer to "examine the process of processing information, while in the process of doing so" (435).

Alternatively the result of an artwork can be seen purely as an artefact of a creative process. As Biggs puts it "creativity is an activity, not a thing, .. In this vision the work of art appears as no more than the dead and decaying remains of what was the living creative activity" (Biggs) . He is referring to the creative process not the process of running software. He also views this with relation to the transient nature of software art which is difficult to collect and preserve. In the case of software art the creative process of writing the code is a way of transcoding the behaviour intended by the artist into a format which is executed on a machine. The machine becomes a performer or the instructions and algorithms of the artist.

The concept can be encoded into the algorithms in the software and into the data which is the content of a work. The algorithms also represent the process as it is the hardware that executes the code or performs the concept as describe in code by the artist. So the concept and the process becomes digitally intertwined

and trans-coded. Bunt explains this merging: "Programming is based on step by step procedures, algorithms. Algorithms can be regarded as sets of instructions that manipulate data. Data represents the dimension of content, while algorithms represent the dimension of process. There is nothing, however, at the lower level that materially or symbolically separates them" (Bunt, "Risking code" 24).

Thayer confirms that his work should be read on a conceptual and a process level "My recent series of Microcodes are intended to be critically examined at the code level as well as at the level of the running process. The code informs the conceptual ideas behind each piece while the running process lends it a more 'poetic air'" ("Microcode Primer").

If the concept and the process are captured in the code and the data which can be distributed because of the principles and characteristics of discussed in the previous section, it means then that software becomes the transmission medium for the concepts and process of the artist. More than just a transmission medium the artist may choose to use a technology to "provoke reflection" in the audience (Trifonova, Jaccheri, and Bergaust 61). In Trifonova, Jaccheri, and Bergaust's view: "An artist might want the software components of the work to be part of the ideas and 'ethics' of the work and not only tools to reach a certain functionality" (56). The work can reflect on it's "status as code" functioning then as "critical meditation on code that is conducted through the mechanisms of code" (Bunt, "Risking code" 40). Amy Alexander states "most non-art software pretends to be neutral and objective technology- devoid of human influence. Software art opens itself up to examination of its human-created biases and its human-experienced influences - so it helps us understand how these factors operate in "normal"

(non-art) software as well" (Blais and Ippolito 24). Alexander is an example of an artist who uses software code to reflect on the non-art code and so uses code in the form of an artwork as carrier for ideas about non-art code. An example of her work is the installation *SVEN*, which uses surveillance equipment and computer vision to detect likely rock stars in the public without them knowing about it. It uses vision recognition software to match members of the public with footage of rock stars (Alexander).

If this is the artist's intention the choice of software as medium carries meaning and communicates a message to the audience. In work such as Thayer's Micro-codes, for example, the meaning is carried on many levels. From the audience point of view the meaning levels include, the choice of medium, the experience of the running code and also on the code reading level. From the artist interfacing to the computer point of view the software is the language that the artist uses to specify how the hardware should behave. So there is a reading and interpreting in the artist computer direction as well.

On the code level software has similarities with the way language transmits meaning. Cramer refers to software code as a "conceptual notation" (2). This view highlights the connection between software art and conceptual art because software is used to record concepts. Carmer views software art as a subset of conceptual art, however he qualifies by making a distinction between software art which is art that exposes it's instructions and what he calls"software-based art" which does not. For Cramer then making the code visible is a requirement for code based art to be categorise as conceptual art (7).

Software described as coded notation highlights the linguistic nature of software and therefore software art. Natural languages as medium carry meaning from humans to humans. Computer languages carry meaning to both humans and computers.

4.4.2 Software and language

Computer languages, even though they are much more formal than so called natural languages, are still semantic constructions which employ structure and element names for the benefit of the human authors and not the machine layer. It is viewed as good programming practice to write source code that is not only correct for the machine but understandable for humans too (Fowler and Beck 56). Although the software is referred to as code it is a semantically ambiguous interface to the unambiguous instructions at the machine level (Cramer and Fuller 150). So the cross over from expression in a medium in which humans are comfortable to a medium where algorithms are captured in an unambiguous way, meanders through layers of interfaces where the natural language world-view and the code world view both come into play. Both a creative process and a software development process will necessarily be influenced by this. Lee views computing as "notoriously linguistic" (34). Bunt agrees as he views programming as a "form of writing" ("Risking code" 31). Simon practises what he calls "creative writing" style of coding rather than a "problem solving style, of writing software" (Ippolito and Simon).

With relation to what Biggs calls digital art, he views both the act of writing and the act of reading to be dynamic. He views artists as having high levels of lit-

eracy to be able to create experiences on a material and symbolic level. There is then an "interplay of reading and writing" on multiple levels. It operates on the hardware level where the code that "materialises the work" is read and written but also the act of reading the work is dynamic as the viewer/participator can interpret it at a symbolic level and alter it by interacting with the work which becomes writing (Biggs). "In these works the explicit processes of 'writing' are as dynamic and motile as their potential 'readings'" (Biggs). Cramer expands on this idea by explaining that the software is the medium in the transmission sense of the word because it can function as sender and receiver but it is also capable of reading and writing (2).

4.4.3 Comparing code and language

The source code is human readable computer commands written in a higher level programming language. It is an abstraction of machine instructions (Krysa and Sedek 237). In the case of digital artworks, the source code is not often made visible to the audience. In cases such as Thayer's Microcodes it is not possible to understand or experience the work without reading the code. Many code based artworks do not make the code visible as it's primary function is to specify the machine behaviour. Even if the source code as such is not read by the audience, the artist writes the code and is dealing with a computer language interspersed with human language when he is creating the code based work.

There are similarities and difference between natural languages and computer languages. Hayles looks at the language world view and compares it to the code world view. She investigates the world-view of print and spoken languages which

she calls human languages and contrasts this with an investigating the world-views of languages which are interpreted or executed by machines. Some of her key points are that the difference between code and language world-view is that code is executed and manifests when it actually runs. A computer system does not allow for ambiguity, so where ambiguity and multiplicity can be created in a literature work, the code version has to be explicit and contain all the possibilities of a narrative. Hayles quotes Barthes "Yet at the same time he can also assert that 'the text must not be understood as a computable object,' 'computable' here meaning limited, finite, bound, able to be reckoned" ("Print Is Flat" 68). So the code is not a text from the computer perspective but the output in the form of either a literary work or an artwork is a text for human participators. Both Thayer and Cramer propose that the source code of the artwork can also function as text for the audience (Cramer 5; Thayer, "Microcodes - Pall Thayer"). In Thayer's case the source code is an integral part of the reading of the work.

4.4.4 Software as unconscious of language

When we communicate using computer mediated tools, our communication passes through layers of software operating with various levels of cognisance on what we say, write, hear or read. Hayles draws the analogy that theses layers of software can be seen as the unconscious of language. She bases this analogy on the fact that the software layers are mostly inaccessible to the users of the software. In a similar way that the unconscious shows itself in the form of a slip of the tongue or a pun, the software becomes visible when it makes unexpected changes such as a predictive text replacement in a text message. In the case of

software artwork the artist communicates her/his ideas through software layers to viewers/participants. Often the software is not made visible to the viewer: "[the software or programming process] is not something that I can show to an audience except in some kind of allegorised, indirect manner" (Appendix III: Email interview with Brogan Bunt) but presenting the code can also be an artistic strategy as in the case of Thayer's Microcodes.

Whether the source code is visible to the audience or not, when the artist is working on a large project, all the source code is not completely visible or understandable to the artist. To deal with this problem software programmers create models and abstractions of the software. Proske's solution to dealing with implementation complexity use abstractions (Appendix V: Skype interview with Pierre Proske). Models can be used to look at a specific area of the software. For instance the software structure can be modelled without looking at the dynamic time based behaviours or the time-based sequences of the software can be modelled while ignoring the structure view. Using models also allow the programmer-artist to construct conceptual models which are not only limited to functional and behavioural aspects of the software, but reflect meaning constructions relating to the conceptual underpinnings of the work. In this way the software medium becomes an environment to explore conceptual models.

4.5 Software as functioning environment

4.5.1 Code and machine as model of reality

To enable programmers and artists to create code they create abstractions to make the underlying code and working of the hardware understandable. Shore

and Warden explain "All these things [design, diagrams] are abstractions - even source code. The reality of software's billions of evanescent electrical charges is inconceivably complex, so we create simplified models that we can understand" (338). These abstractions are models of how the software works as well as models of aspects of reality as experienced by the coder/artist. In Bunt's view abstraction and model making is used to decompose all elements of a system, it's states and processes into binary states and logical operations so that they can be represented in logical and symbolic form (Bunt, "Risking code " xx). So model making and abstraction is a fundamental part of creating code and is a basic activity when coding software. In Manovich's view reality can be modelled by software in two basic ways: by algorithm or by data structures ("New Media"). This applies to software art as well. If we see the algorithmic aspects of the code as the codification of the concept and process and the data and its related structure as the coded from of the content, software can then be seen as a model of reality through the artists' concepts.

4.5.2 Software as muse

In some cases writing code becomes the driving force and is no longer only a way to solve an artistic problem. "For many practitioners, code is not simply a means to an end; on the contrary, they revel in the intricacies of document.write; they chisel lines of Perl or Java instead of marble, creating elegant solutions to artistic problems. Code is their muse" (Blais and Ippolito 17). In this case the process of writing the code is the focus and goal of the creative activity. Even if the process is the muse of the artist, it is not always possible or useful to separate the

process and the concepts. Software as muse can apply to both the code and the execution. In Bunt's view "Code and execution are tied together, neither subordinate to the other. If anything, in its silence and disappearance, the plane of execution provides the well of darkness from which the potential for creative conceptualisation emerges" (Bunt, "Computational Drawing : Code and Invisible Operation" 1). For Bunt the execution becomes the inspiration or muse for creative conceptualisation.

4.5.3 Software as metaphor

According to the free dictionary a metaphor is used to make implicit comparisons by using a phrase or word which ordinarily designates one thing to designate another ("metaphor"). So one thing, idea or concept is used to give information about another by comparison. Software is used as a metaphor for "mind, for culture, for ideology, for biology and for the economy" (Chun 2). Usually a metaphor uses a known entity to compare to an unknown entity. Because it is known it provides clarity about something unknown. Software is described as "almost intangible, generally invisible, complex, vast and difficult to comprehend" by computer scientist Manfred Boyd (Chun 3). In the case of software as metaphor it is the fact that software is unknowable which is used as the metaphor. This paradox makes software suitable to to explain "something invisible that generates visible effects" (Chun 17).

If artists use software but choose not to reveal the underlying code the choice of medium echoes the metaphor of something unknowable causing visible effects. Even if the artist chooses to bring the code of an artwork to the forefront

there are still layers which are invisible and unknowable. Making the code visible may also enhance the viewers experience of the metaphor because, even to experienced coders, the working of the code may not be obvious and may require an explanation. The visibility of the code and the explanation highlights again the esoteric nature of the code to coders and non-coders alike.

4.6 Software as end product

To say the medium of an artwork is software in the intervening substance sense of the word or in the artistic technique sense of the word is true but a simplification. The medium "is simply something that occurs in between and can occur at any point between the artist and the viewer" (Thayer, "On artists"). Programming becomes the "creative conduit" (Appendix VII: Email interview with Pall Thayer) but also the muse, metaphor and model. It is the environment in which the concepts of the artist are performed/executed. It is the text which the artist, machine and viewer/reader can read and write. The multiple meaning of the word medium is applicable to the different approaches to software art. Software can function as a tool, as a technique or as a material for an artwork. In this role characteristics such as programmability, mutability, re-mixability, flexibility, non-static nature and linguistic nature has been identified. Software can provide a transmission medium to communicate ideas and concepts. In this approach it is used to carry meanings and has similarities with conceptual art. It can also be compared to language. Software can be used as a medium in which the ideas or concepts of an artist can grow. This approach highlights the use of software as muse, as metaphor and as simulation environment. Lastly software can be the

medium of the artwork but it can also be end-product. This overview of the approaches to software art, which I have found in my reading, provides the background for a better understanding of code based artwork to inform the discussion and comparisons of the next chapter.

5 Discussion

5.1 Research questions and how they relate to the areas of study

The research questions, as posed in the introduction, are:

How do practising software artist experience their development process? How does this process compare with the Agile software development process? How can conclusions made from a comparison between the Agile process and the discussions held with practising software artists shed light on the areas where the Agile process can assist artists and areas which might be avoided?

The research questions require all the preceding chapters in this writing to support the answers. A map of the comparisons which are required to answer the questions and the section in which each question is answered is presented in fig. 8.

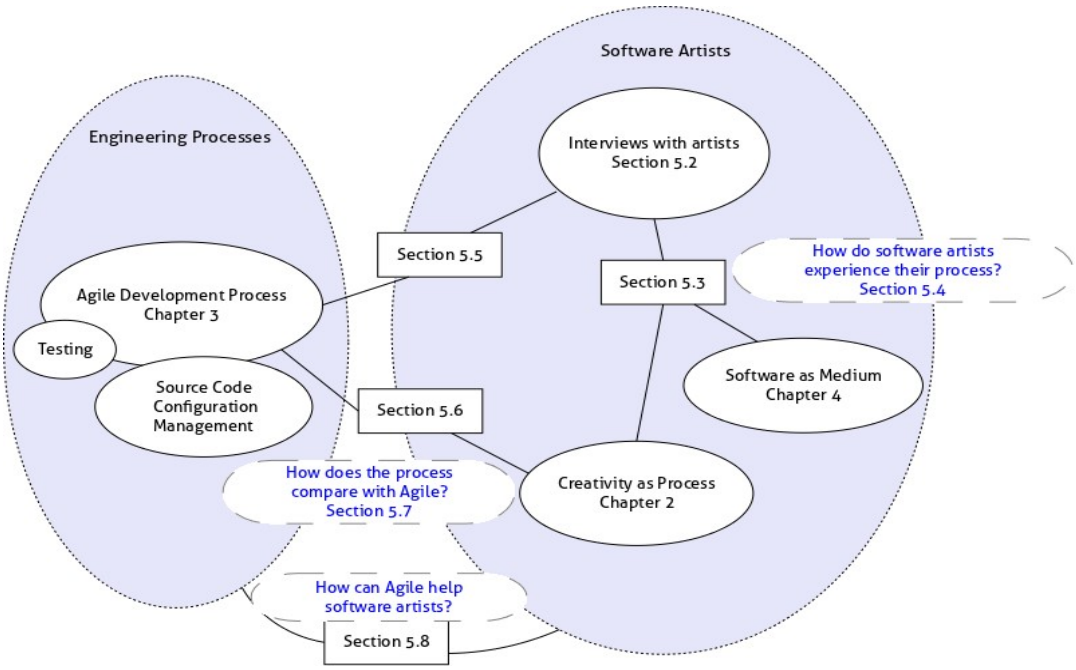


Fig. 8: Map of chapter and section relationships

To answer the first question "How do practising software artist experience their development process?" interviews were conducted with artists using email and Skype. The interviews are presented in section 5.2. To make sense of the artists' experiences they have to be seen relative to the literature study on using software as medium as described in chapter 4 as well as the literature study on the creative process as described in chapter 2. The interview results in relation to software as medium and the creative process are discussed in section 5.3. The answer to this question is discussed in section 5.4.

To answer the question "How does this process compare with the Agile software development process?" the ideas uncovered in the interviews are viewed from an Agile perspective in section 5.5. Creating a software artwork is a software development process as well as a creative process. Section 5.6 discusses Agile process elements as discussed in chapter 3 and compares them to aspects of the creative process as highlighted in chapter 2. The answer to the question will be discussed in section 5.7.

Based on the discussions of the different areas the final question "How can conclusions made from a comparison between the Agile process and the discussions held with practising software artists shed light on the areas where the Agile process can assist artists and areas which might be avoided?" will be answered in section 5.8.

5.2 Interviews with Artists

Personal interviews, using email and voice, were conducted with 5 artists. The interviewees were asked to answer 3 questions:

1. Has using software as medium changed your creative process?
2. If so how?
3. What problems or benefits do you experience when creating an artwork using software?

Where necessary interviews were supplemented by artists statements and documentation about artworks and exhibitions.

5.2.1 Brogan Bunt

Brogan Bunt is a media artist and an academic. He creates video and software art work. He is the Head of the School of Art and Design, University of Wollongong, NSW, Australia. He has also written about "aesthetic issues emerging from the contemporary effort to position programming as a form of artistic practice" ("About | brogan bunt"). His work includes interactive software based work but also custom software tools. Bunt views the practice of coding software to be very similar to his creative process, which involves rules and systems. He feels the process of programming is only important to him but not to his audience and that he can only show the programming aspects indirectly. In a recent work, Loom, the code is only made visible in the resulting visual patterns (Appendix III: Email interview with Brogan Bunt). Bunt is exploring links to traditions of instruction-based conceptual art. He believes digital artworks have a dual nature: a conceptual nature and a machine executed process nature. For him the relationship between the two aspects are no longer binary or hierarchical but enmeshed and congruent ("Computational Drawing"). Images from *Loom* are included in Fig. 9 and Fig. 10. The images in printed form were the artefacts which were presented

to the audience. Bunt's code is not visible and is only indirectly visible in the geometric variations presented in the imagery. Bunt refers to the work of Sol le Witt in the opening discussion of this work. Le Witt focusses on the concept of his drawings rather than the actual drawing. Bunt's work investigates the conceptual logic of the drawing and contrasts it with the execution of the logic by the computer in an electronic space. In these images polygons are sub-divided according to rules programmed by the artist, much like le Witt's instructions for a draughtsperson explaining how to create the drawings. Bunt engages with the concept of labour where he looks at the difference between programming labour as done by the coder and execution labour as performed by the computer. Human reason is contrasted with the repetitive performance of machine execution. The title of the series, *Loom,* refers to the Jacquard loom which was used to weave cloth according to algorithmic patters. The Jacquard loom was used to weave complex textile patters and replaced human weavers in the rise of industrial manufacturing processes. So the visibility of the artefact, the image and the invisibility of the process that created the artefact relates to the concept of how we value different classes of labour. In his paper about this work, Bunt contrasts this distinction of labour with the way that Aboriginal paintings are created with both repetitive hatching and figurative elements. The repetitive hatching is not performed by the same person that adds the figurative elements. The hatching is regarded as a "repetitive articulation of time and space" and the ritual nature of the activity performs the role of "summoning and invocation" of an ancestral being, therefore viewing this activity as important and integral to the painting, as important as the figurative elements ("Computational Drawing").

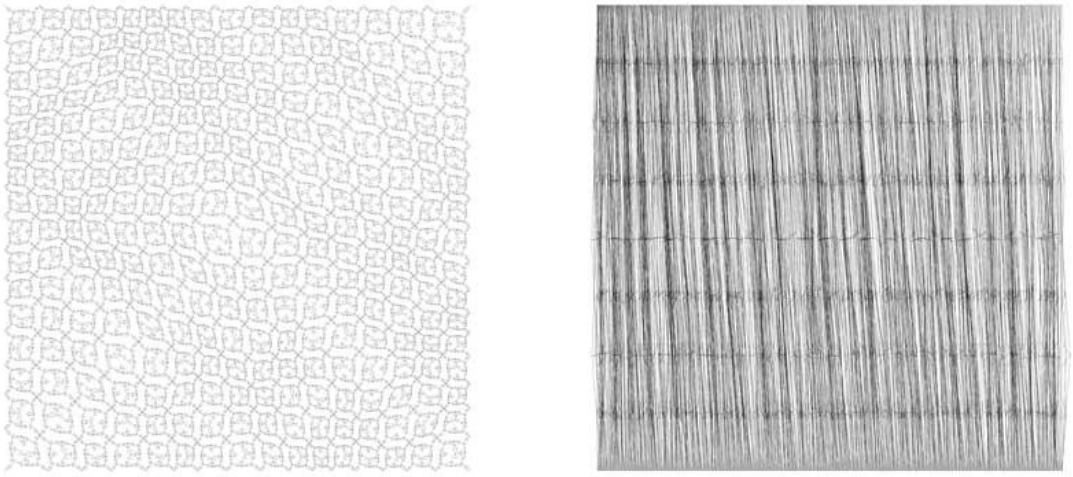


Fig. 9: Image 5 and 6 of the series Loom Bunt (2011)

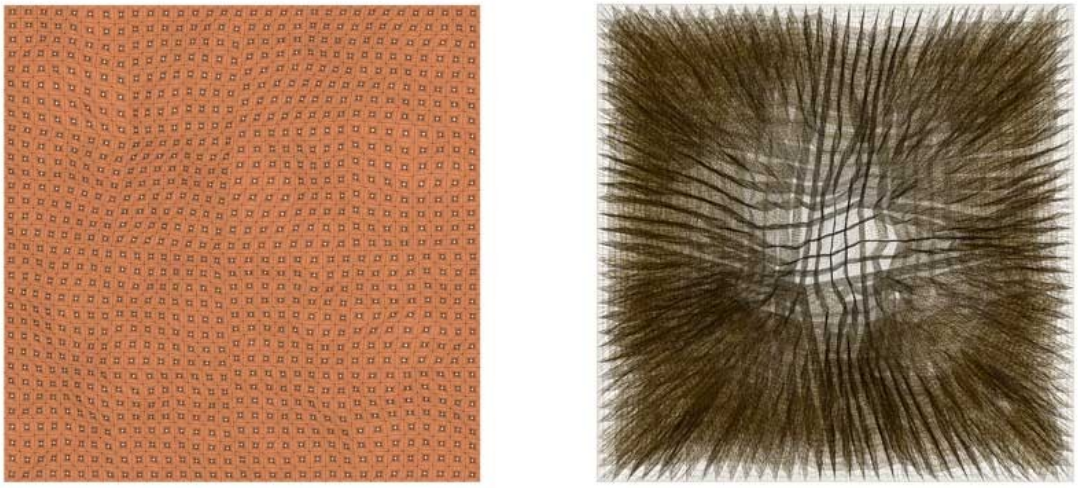


Fig. 10: Image 14 and 15 of the series Loom Bunt(2011)

5.2.2 Joshua Goldberg

Joshua Goldberg is a New York based artist who uses software systems to perform custom sound visualisations. He calls himself a 'live visualist' ("joshuagoldberg"). He also writes custom software tools which he has released to the public.

The user interface of his custom software tool, *Dervish*, is shown in Fig. 11. Goldberg believes it is the digital artist's responsibility to push the boundaries of the tools and technology he uses. For digital work to be viable, relevant and interesting, the artist must be aware of the limitations of the tools and must transcend "the limitations and the intention of that tool to push the envelope further" (Appendix IV: Transcription of Skype Interview with Joshua Goldberg). He believes that an artist is not making art if he is not pushing the boundaries. It is for this reason that he created software tools to create the visual effects that he uses in his performances. His custom tools provides him with a mechanism to push the software and the hardware to its boundaries and removing limitations which may be caused by third party tool implementations. Fig. 12 and Fig. 13 show frames from a video created to show the effects of the tool *Dervish (Goldberg, "Dervish-meditation 2012 on Vimeo")*. Goldberg performs interactive animations using the tools he creates.

Goldberg also says the artist should be aware of the transience of the medium. He says "You're making art for the decade, you are not making art for the ages". An example of this is file formats which will no longer be supported. This means work, that is created in a file format which becomes obsolete, will no longer be accessible to audiences once the format is no longer supported. Another example of the transience and limitations of the medium is an artwork which has specific visual characteristics because of an interaction with the frequency of the supplied power. This means the work can only be shown where a 110Hz power supply is available.

In Goldberg's view engineering process can be applied as artistic processes if it is a "coherent artistic strategy" and if it is then framed as an artwork. He does feel there is a difference in approach between engineers and artists. In his opinion the difference lies in the responsibility of the engineer and the artist. He says: "it's not the artist's responsibility to get it perfect just to think about new viewpoints, new ideas, new comments, new work-flows, whereas the engineers responsibility, true responsibility is you have to get it perfect" (Appendix IV: Transcription of Skype Interview with Joshua Goldberg).

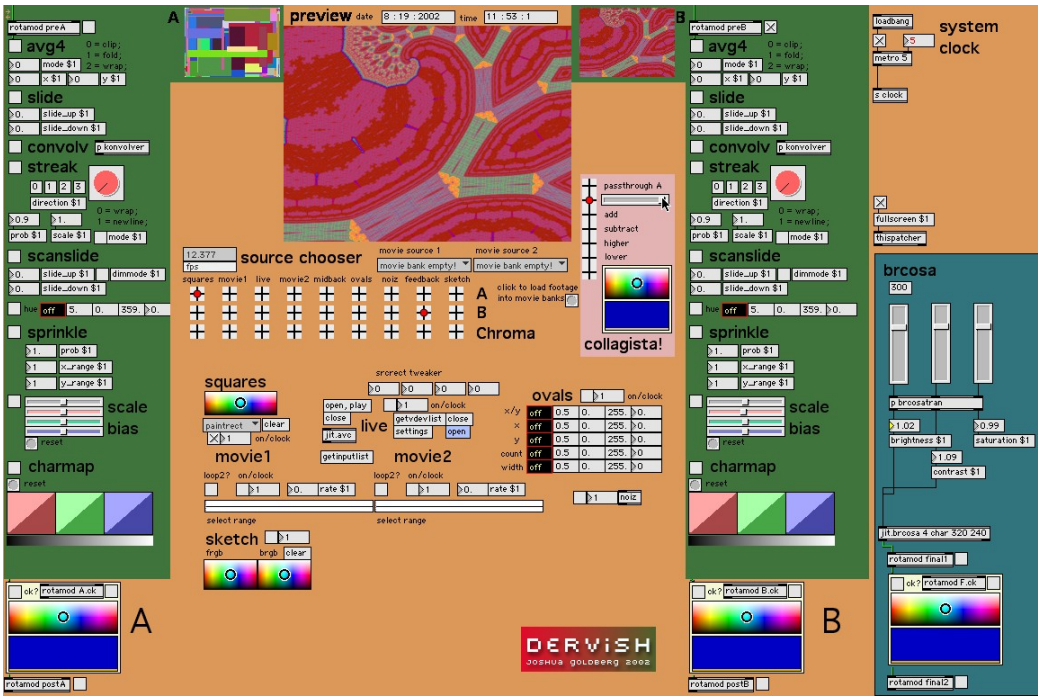


Fig. 11: User interface of the program Dervish Goldberg (2002)

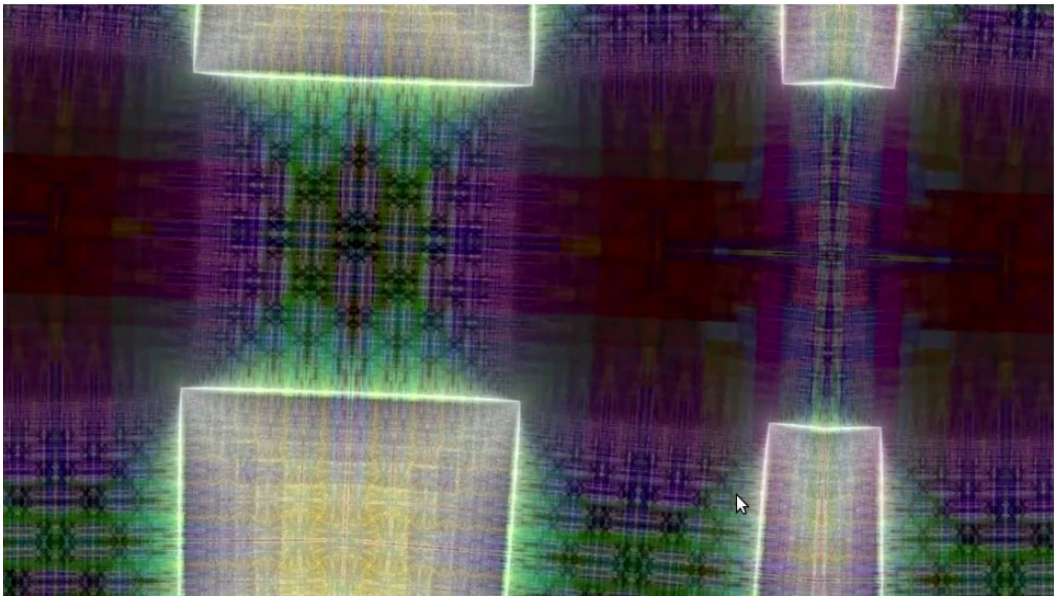


Fig. 12: Frame from the video dervish meditation - 2012 Goldberg (2012)

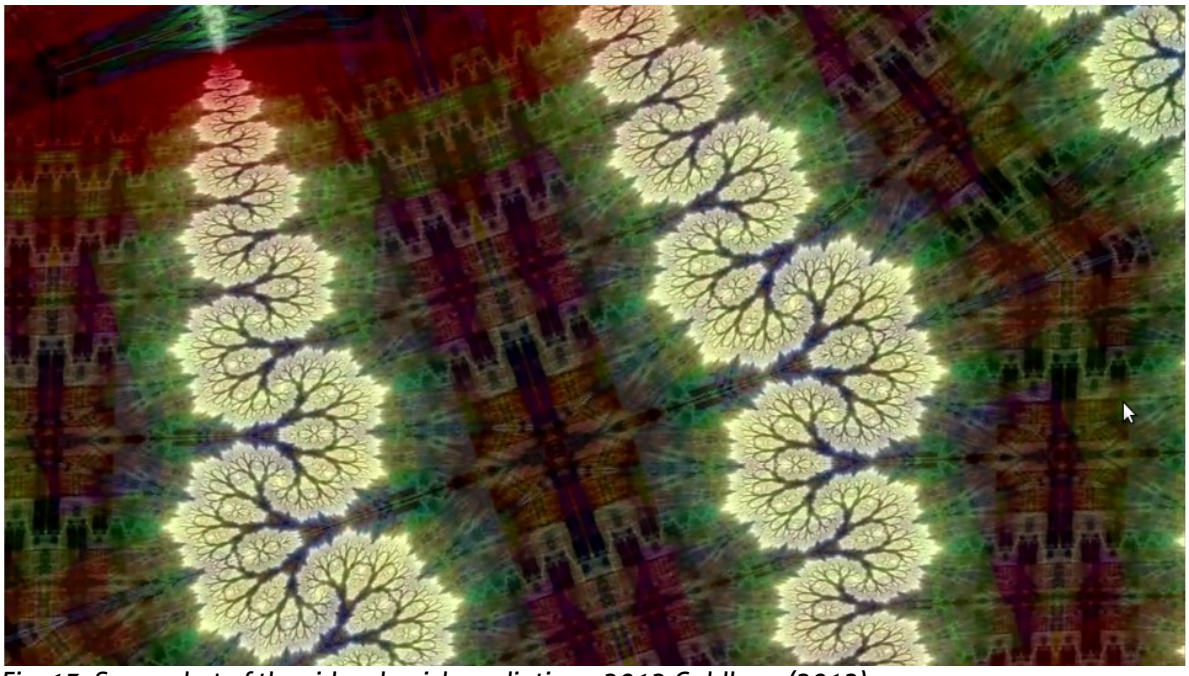


Fig. 13: Screenshot of the video dervish mediation - 2012 Goldberg (2012)

5.2.3 Pierre Proske

Pierre Proske is an Australian artist who merges his parallel interest in technology and the arts. He studied electrical engineering and liberal arts at undergraduate level and completed a Masters in Art and Technology at Chalmers university in

Sweden. He has worked as a sound designer and electronic musician and has exhibited or performed in Australia, Sweden, Canada, Iceland, Brazil, Japan, Austria and the Netherlands (Proske).

Proske comes from a music and electronic music creative background. Earlier in his career he was focussing on making content and he did not want to become involved in making tools. He felt making tools distracted him from his creative goals. This evolved to his present process which involves creating most elements of his work himself. His distinction between art work and tool has become increasingly blurred. He feels he should maintain a balance between making tools and making art work.

This shift, according to him, is because, in terms of making consumable electronic music, it made sense to be content driven. However shifting into a contemporary art framing he feels the concepts became more important. Also he thinks, to make something original he has to be close to the coding process. To get a product that is "uniquely yours" an artist needs to be engaged in the medium and therefore close to the coding process. He thinks though that to be engaged in the medium requires a level of technical skill and this excludes many people from taking part in the practice.

On discussing the difference between the coding process for art or non-art purposes, Proske says art projects have more liberty and less rigid requirements. There is also a more playful element to this process. The coding practice, for him, is very powerful but the process can cause him to "get lost in the implementation details" with a result that "whatever sparked the project initially would fall by the wayside". This occurred especially when developing large projects. In this regard

re-using coding tool-kits are important for him because it allows him to keep focussed on his conceptual goals because large re-usable parts of the project have already been implemented.

A way for him to manage the complexity is through methods of abstraction. He says "the more you can abstract out the complexity...the more you can connect to your actual concepts".

The software development aspects of the coding process he sees as "fast and dirty" but he relishes this mode of working. He sees source code control as an essential mechanism for documenting his work. He feels documenting your work in an art context is very important. So he says if code is the work this means source control is a self documenting tool. He also thinks the ability to branch and roll back your code to a specific point is critical to his creative exploring process.

He thinks the amount of testing is often scant but that it depends on the longevity of a project and its exposure to the public. If at some point code artwork becomes part of the mainstream collectible art market, then he thinks artists may need to employ more rigorous engineering processes to create more robust works.

He sees the medium as transient and this is another reason he stresses the importance of source control as a recording mechanism because it provides a way to retrieve code so that it can be ported if the running platform becomes obsolete. He thinks the transience may be reduced as the medium becomes more mainstream. He also sees transience as playing an active role in the creative process because he says in any medium, an artists must be critical when refining a work.

From his engineering studies he described a shift in approach which he feels defined him. He thinks he needed to raise philosophical questions asking why things are done and he did not experience this kind of questioning approach in his peers. He thinks lateral creative thinking approaches can be useful in terms of achieving innovative solutions. The transcript of the interview is recorded in Appendix V: Skype interview with Pierre Proske.

Proske collaborates with artists and creates interactive installations. An example of his work is the software work titled *Abstract Microecologies*. This work is an algorithmic software system which creates collages out of imagery that was collected during a residency at the Department of Archaeology and Natural History at the Australian National University, Canberra, Australia. The images are microscopic images of fossilised pollen and micro organisms. The algorithmic animation of the images create a cloud-like interaction of the particles with each other which reminds the viewer of clouds of pollen floating in the air. In this way the particles which are static and fossilised can be seen in a dynamic interaction. This work was presented as print and as video. This work refers to "systems which generate ecologies of interrelated, distributed elements" in both the science and the art fields. This also mirrors the software code and data structures which would be necessary to create the visuals of algorithmically changing animations which render multiple images (Proske, "Abstract Microecologies"). Screen captures of this work can be seen in Fig. 14, Fig. 15 and Fig 16.

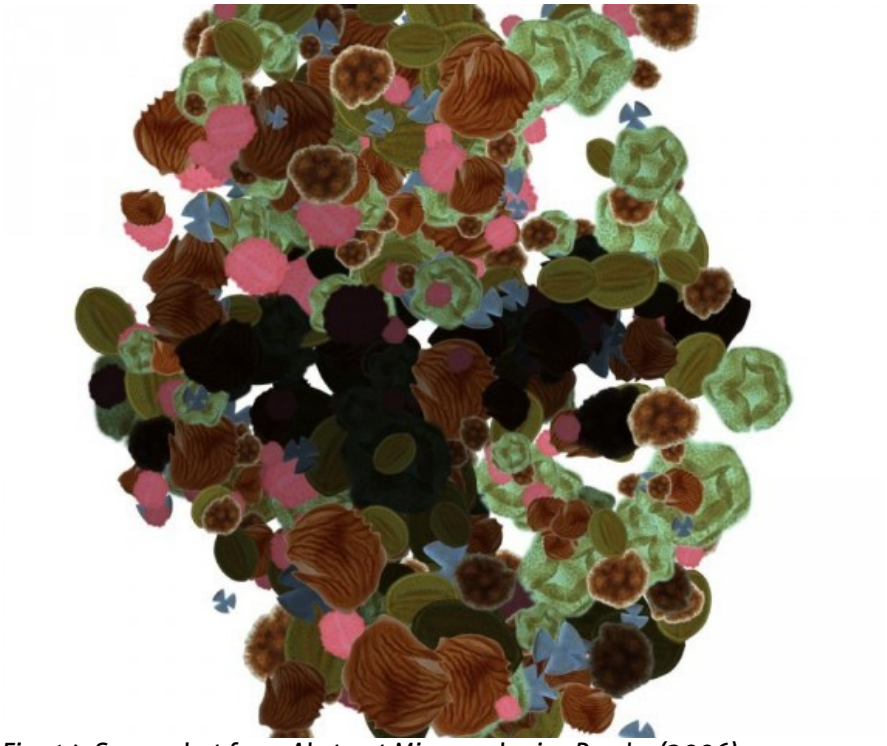


Fig. 14: Screenshot from Abstract Microecologies Proske (2006)



Fig. 15: Screenshot of a video, Abstract Microecologies Proske (2006)



Fig 16: Screenshot of video, Abstract Microecologies Proske (2006)

5.2.4 Nathaniel Stern

Nathaniel Stern is an installation, video and Internet artist. He is based in USA and South Africa. He uses both traditional media and technology to create his work. He teaches at the Department of Art and Design at the University of Wisconsin and writes about art (Stern, "nathaniel stern : short artist biography"). Like any tool, using software affects/effects his thinking and his work. If works that use different tools are compared to each other, he says these effects are visible. He says that "the more one learns about varying tools, the more their options fan out, and they can start thinking across various modes of making, then worry about the language/code/software to use later" (Appendix VI: Email interview with Nathaniel Stern). Knowledge of a particular tool enables him to choose the appropriate tool to express his ideas. His later work focuses on the relationships between personal or professional, online or offline, artists and the academy, epistemology and technology, bodies and space, or history and public dialogue. He uses technology to allow viewers to explore these relationships (Stern, "nathaniel stern : artist statement and general interests"). His choice of technology as medium allows him to explore options and to highlight different relationships.

5.2.5 Pall Thayer

Pall Thayer is and artist and lecturer at SUNY Purchase College, NY, USA. He is based in Iceland and the USA (Thayer, "Pall Thayer - artist"). He created a series of small code based artworks, called *Microcodes*, where the meanings are deciphered from a combination of the title of the work, reading the code and running the code (Thayer, "Microcodes - Pall Thayer"). The website presenting the

Microcodes is seen in Fig. 17. The Thayer was exploring abstraction as strategy before he chose code as medium. In his exploration the abstraction was separating the conceptual space from the execution or the creative act. To start using code as medium was a useful progression because it gave him access to distributed audiences on the Internet. He considers the code he creates and not the user interactions as his art. He feels that his work *On Everything* was misinterpreted because people were not looking at the code of the work. As a response he created Microcodes which can not be viewed or understood without reading the code.

For Thayer creating software, not using software has changed his creative process. He feels there is an important difference between using software and creating software. He feels that artists who do not code their own software are not using code as medium. Artists create works using code as medium but the audience experiences the work in a different medium. This causes a divide between the medium of the artist and the medium of the viewer. He thinks therefore it is important to make his audience aware of the code (Appendix VII: Email interview with Pall Thayer).

In the work *Microcodes* each code "poem" can be read. The code is presented to the viewer on the website. In addition the code can be pasted into a file and executed if a suitable perl interpreter is present on the system. The running code may or may not produce an output on the screen. In addition the title of each piece provides clues about the concept of the work. A selection of three Microcodes are presented here. First the code is presented, then the view of the code in a text editor on a typical viewer computer and the output of the running script is presented.

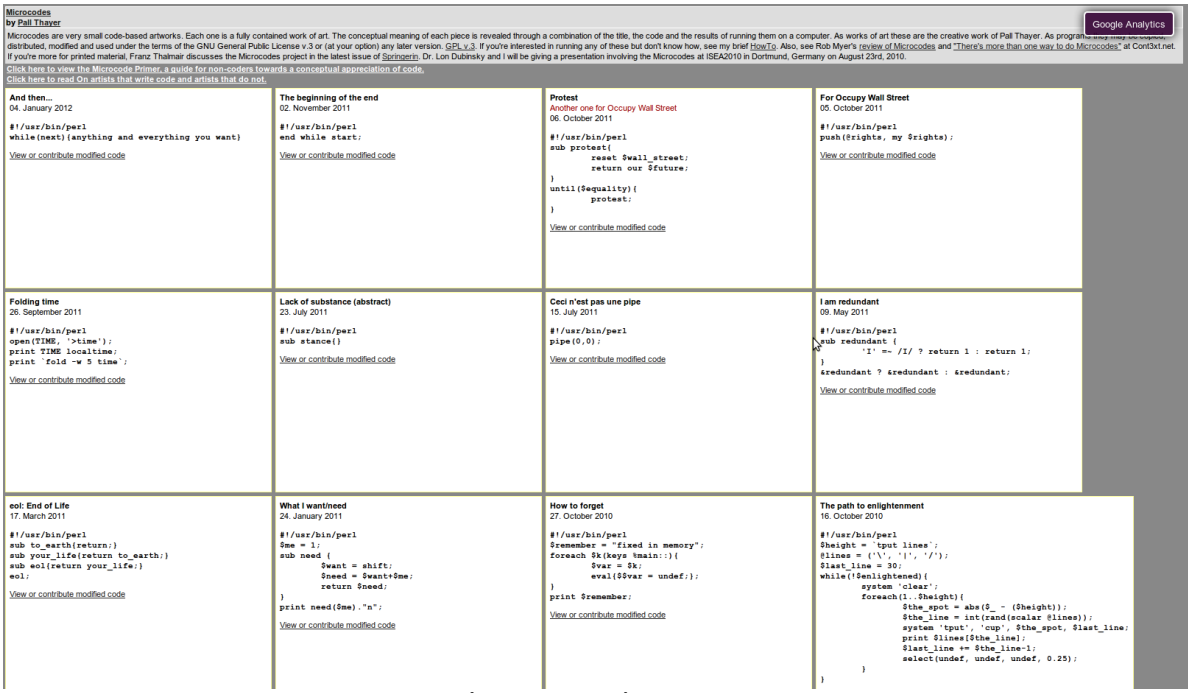


Fig. 17: Microcodes website Pall Thayer (2009 - 2012)

An Icelandic landscape

14. March 2009

```
#!/usr/bin/perl
use LWP::Simple;
@data = get("http://www.google.com/ig/api?weather=Reykjavik") =~ /(d+)/g;
@char = ('*','.','#');
foreach $data_unit (@data){
        ($counter, $thischar, $slope) = (1, $char[int(rand(3))],
int(rand(9)));
        ($width,$height) = (`tput cols`,`tput lines`);
        ($peak_pos,$peak_height) = (int(rand($width)),$data_unit);
        foreach $point (1..$height){
                if($point>$peak_height){
                        last if int($width-$peak_pos-int($counter/2))<0;
                        system 'tput','cup',$point,int($width-$peak_pos-
int($counter/2));
                        print "$thischar"x$counter;$counter = $counter+
$slope;
                }
        }
}
```

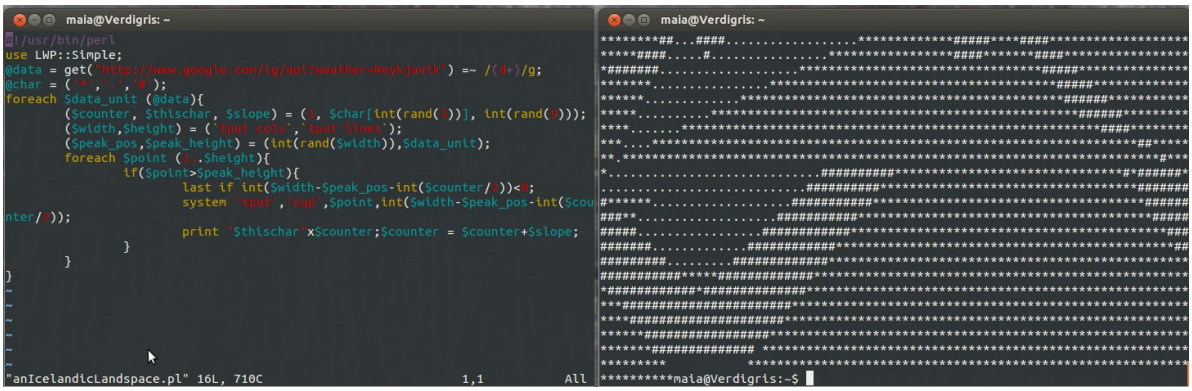


Fig. 18: an Icelandic Landscape, code and output Pall Thayer (2009)

In this work, the code takes weather information for Reykjavik, the capital of Iceland, from the google api and draws a series of peaks, slopes and points thus creating a landscape based on current weather conditions in Iceland. This view of the artwork is only possible by reading the code. The code informs the visuals that are created. The title of the work is *an Icelandic Landscape*. This title also refers to the roots of contemporary art in Iceland, where early contemporary painting presented landscape imagery of Iceland ("National Gallery of Iceland").

White on white

Results may vary depending on terminals 26. March 2009

```
#!/usr/bin/perl
system 'tput', 'setb', '7';system 'clear';
($height, $width) = (`tput lines`-10, `tput cols`-10);
print "33[37mn" x 5;
foreach $line (1..$height){
        print " " x 5;
        print "*" x $width;
        print "n";
}
print "n33[0m" x 5;
```

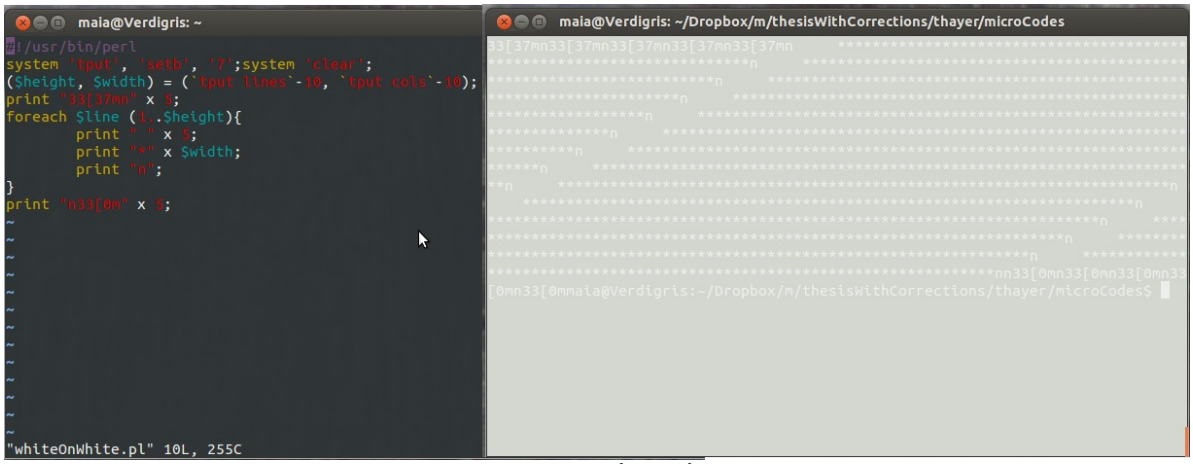


Fig. 19: white on white, code and output Pall Thayer (2009)

This work takes the height and width of the terminal window in which it is run into consideration. It changes the background colour to a greyish white and fills the terminal window with text asterisks and spaces. The algorithm also creates diagonal blank spaces. This work refers to work by the Russian artist Kazimir Malevich with the same title. Malevich's painting is a covered in brush work in shade of white with a rectangle at an angle painted in a different shade of white ("MoMA | The Collection | Kazimir Malevich. Suprematist Composition: White on White. 1918"). The visual output of this microcode and the title provides the clues to the art-historical references that Thayer is making.

CNN Dada

```
#!/usr/bin/perl
use LWP::Simple;
system 'clear';
$width = `tput cols`;
$height = `tput lines`;
$text = get('http://rss.cnn.com/rss/edition.rss');
$text =~ s/<[^>]*?>//g;
@words = split(' ', $text);
while(1){
        ($horiz, $vert) = (int(rand($width)), int(rand($height)));
        system 'tput', 'cup', ($vert, $horiz);
        print $words[int(rand(scalar @words))];
}
```

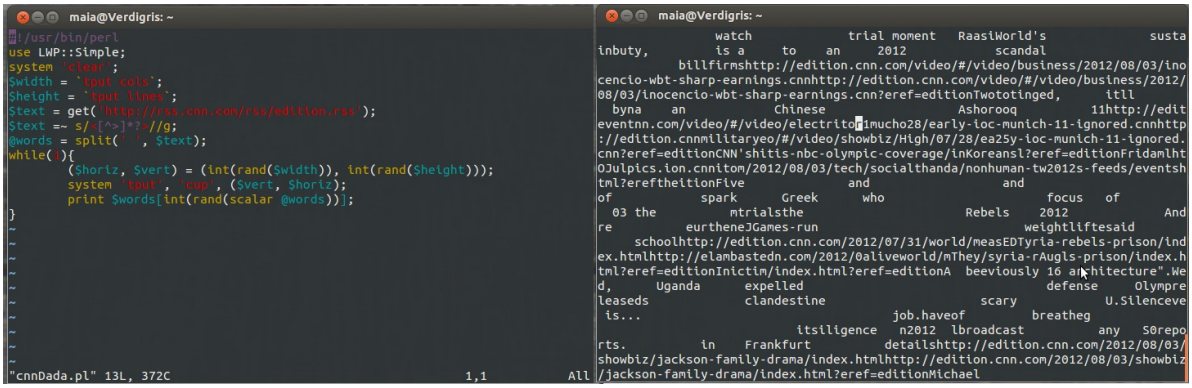


Fig. 20: cnn Dada, code and output Pall Thayer (2009)

Here Thayer takes text from the news feed of the CNN website. The text is split into pieces and then randomly placed on the screen. Each time the script is run it produces different output. This work refers to the Dada technique where the artist takes ready made objects or text from everyday objects and creates artwork by recombining the fragments in a collage. An example of a Dada artist which worked in this way is Kurt Schwitters. His work titled *Mai 191* created in 1919 ("Small Sailors' Home - Kurt Schwitters – WikiPaintings.org") was created using a collage of fragments from newspapers..The title, the code and the output adds to the conceptual reference that Thayer is making in this work. The code is integral in understanding the conceptual elements of this work.

For Thayer visible code and how it works is of primary importance. Even if the viewer is not fluent in Perl, it is still possible to discern concepts and words such as the websites which are accessed and the variable names such as slope, peak, height and width. The code snippets are short which also aids the viewer in accessing the code. The title of each work is crucial in understanding the reference that the microcodes make. The visual output may not be accessible to all viewers but provides a sense of discovery when the code is run and the output is re-

vealed. It also validates the code snippets as code because they are syntactically correct and executable by the computer.

5.2.6 Comparison of interviews

These five artists were chosen because they are exhibiting artists and because they wrote code as part of their artworks. This means that all five artists experienced creating code for an artwork as part of their practice. There are recurring themes that more than one artist mentioned in the interviews and in the artist statements. The themes I identified are:

1. comments on software as medium
2. comments on software as tool
3. comments on the conceptual nature of software art
4. the visibility of the code in an artwork
5. aspects of the engineering approach.

An analysis of the interviews show process activities that are important to each artist. These activities and approaches can be related to the Agile process. Artists describe their process in terms of their experience. What each artist highlights provides a way of seeing what the artists sees as an important aspect of the process.

Software was seen as a transient medium. It was also seen as a medium that is not necessarily the same for the artist and for the audience. Two artists found that the transition to software as a medium was natural as their creative process had similarities with code writing activities. In Bunt's case this was because his process is rule-based and in Thayer's case this was because his process relies on ab-

straction. Proske sees software code as a medium with which he must engage and code to create his work. Thayer agrees that creating code, not using code changed his process, that is, using software as medium and not as tool affected his process.

Software as tool was seen to change the making and thinking process for Stern. Goldberg thinks the limitations of software tools should be known and transcended to make relevant art. Proske thinks that making software tools can be a distraction from the conceptual elements of his work but that it is necessary to code his own tools to create what he intends. He thinks it is important to maintain a balance between making tools and making art.

Stern thinks that having knowledge of the tools allows him to conceive different making possibilities and conceptual elements without needing to resolve implementation details such as code or tool choices as part of the initial process. Proske also thinks the conceptual elements of his work is important and he thinks that complex implementation detail can distract him from the conceptual core of his work. He counters the complexity with re-use of software modules and abstraction. Thayer also uses abstraction to separate conceptual elements from execution of the work. On the other hand Bunt thinks that the conceptual nature and the execution nature of his work are enmeshed.

On a conceptual level Thayer feels it is important to make the code of his work visible. By contrast Bunt shows the code of his work indirectly in the artefacts he creates. Goldberg has made some of his code available for download as has Proske, which effectively makes the code visible. This visibility is not part of the artwork as in Thayer's case.

Goldberg and Proske agree that engineering processes are part of the software creation process but both these artists feel that the approach, which engineers use, differs from their approach. The difference both artists gave was that artists ask questions and engineers build solutions. None of the other artists commented on engineering processes or approaches.

The approach to goals and the goals of a project play a big role in the way the development and creative processes occur because it defines what the priorities are, affects the cognitive style and therefore what kind of cognitive activities will occur. To explore the way artists and technologist describe their different approaches and to supplement the interviews, an editorial article in Rhizome.org provided a varied view ("Rhizome | Do Artists").

5.2.7 Opinions of artists and technologists

Rhizome is the organization founded by Mark Tribe that supports emerging artistic practices. Rhizome has an event called *seven on seven*. This event groups seven artists and seven technologists in groups of two to create something new over the course of a day ("Seven on Seven - Rhizome"). An editorial article in Rhizome asked the participants of *seven on seven* to respond to the question: "Do Artists and Technologists Create Things the Same Way?" ("Rhizome | Do Artists"). This discussion presents opinions from both artists and technologists about their creative process and reveals views on the differences, particularly in the goals and approaches. Michael Bell-Smith states that he thinks "both put a premium on new ideas" but that the difference is in the goal because he thinks "good technology is about solving problems, while good art is about creating problems". Kellan

Elliot-McCrea explains that "engineers focus on removing ambiguity and artist often take the opposite approach". It is a generalisation to attribute a specific approach to a group of people. Also to belong to one group does not preclude belonging the other, "a 'technologist' could easily identify an situate themselves as an artist, it is a choice of field and context" according to Emily Roysdon ("Rhizome | Do Artists"). By contrast Zachery Lieberman thinks there is no difference between artists and technologists because both ask questions and search for solutions in a creative way ("Rhizome | Do Artists").

The ideas presented in this discussion are collated. The categorisation of technologist/artist is a generalisation and people can be a member of both categories at the same time. Both technologist and artists put a premium on new ideas. There may be a difference in goals: artists create problems and technologists solve problems. There may also be a difference in approaches: artist try to create ambiguity, technologists try to remove ambiguity. There may be no difference as both ask questions and search for solutions in a creative way. What is useful however is to notice that there are different approaches which result in a different way of dealing with project goals and have a direct impact on the creative process.

5.3 Software as medium, the creative process and interview results

The artists who were interviewed provided varied opinions on topics relating to software as medium. Of the five themes that were identified in the interviews, four are related to software as medium. From this it is clear that software as medium plays an integral role in understanding issues relating to creating software

art projects. In the chapter about software as medium characteristics of the medium were highlighted. One of the points discussed in this chapter was the use of software as a tool. This is also one of the themes identified in the interviews. None of the interviewed artist used the software purely as a tool to create an effect. This may be because the artists were chosen because they have been involved in coding artworks. This means that the artists who were interviewed used the programmable characteristic of the medium as part of their process. Another characteristics of the medium that was mentioned was flexibility. This was particularly important to Thayer. Proske mentions the characteristic re-use which he sees as part of tool-kits as a way to balance making work and making tools. The libraries that he re-uses provides functionality to his work without requiring complex implementation from him. Goldberg mentioned using algorithms for some his work so the programmable characteristic and the capability of running algorithms is important for this artist. The different characteristics of software as medium are used by the interviewed artists.

The interviews also highlighted that these artists engage with the medium on a conceptual level. This confirms that the medium also functions as a transmission medium for the ideas of the artists. Proske stated that the implementation sometimes distracts him from the conceptual aspects of his work. The implementation can be seen as a process that is part of the engineering processes so implementation complexity can be alleviated with engineering solutions such as re-use and abstraction. Proske confirms that this is part of the way he works. This means then that the engineering process is used to help the artist to focus on the area which interest him: the conceptual aspect of his work.

From a creative process point of view, we see that the definitions of creativity concur in that something novel is created. This is important to Goldberg because he says the tools need to be pushed beyond their limitations. Creating something novel is also important to Proske because he feels the only way he can create something unique is by engaging with the medium by coding the software.

Proske refers to convergent and divergent phases of creativity indirectly because he describes making decisions to throw away code to refine work and he describes a process of experimentation and play where art projects have fewer limitations. From his descriptions we can see that elements of the creative process as described in chapter 2 are accurate descriptions of this artists process. This confirms the relevance of the recommendations described in chapter 2 at least for some of the artist that were interviewed.

Another aspect of the creative process which is visible particularly in the *seven on seven* discussions in section 5.2.7 but also surfaces in the interviews is the approach and goal of artists. Goldberg, Proske, Elliot-McCrea and Bell-Smith see a difference in the artist approach compared to the technologist or engineering approach. Even though Lieberman feels there is no difference in approach, the identification of a goal oriented and an exploratory approach is useful as it supports the recommendation that both approaches should be accommodated in a process.

Candy and Edmonds describes different cognitive styles used in collaborative creative projects. One of the facets of these styles is the approach adopted by the team members, either goal oriented or exploratory. Goal oriented means that a specific goal is set at the beginning of the project and the work is done towards

the goal with minor deviation. Exploratory in this context means ideas are generated from details in the process of the project until a result is found (Candy and Edmonds 136). Intuitively the exploratory approach seems to be the way to discover novel ideas. But Candy and Edmonds feel that both approaches have validity and having insight into a different approach, being able to see when an approach can benefit a project even if it is not the natural approach of the team member, is a useful skill (Candy and Edmonds 140).

5.4 How do practising software artist experience their development process?

It is clear from the interviews that many of the issues relating to software as medium were discussed and creative process and engineering process topics were only referenced indirectly. This highlights the importance of topics such as the characteristics of the medium and the conceptual nature of the medium above process details for the interviewees. To answer the question "How do practising software artist experience their development process?": some artists experience that the coding process has similarities with their creative process. However the interviewed artists focussed more on software medium and conceptual themes and did not discuss the development process in detail. The development process is seen as supporting, but sometimes distracting from, the conceptual and creative practice which is their focus.

5.5 The Agile process and interview results

The process of writing software, at the very least, requires some form of implementation phase and some activities to see if the software performs as expected. Since the artists were chose because they write software as part of their art mak-

ing, they share the experience of implementation and evaluation of the software that was implemented even if their approach, process and results differ.

In the interview with Goldberg he was clear that he thinks it is important for an artist working in the digital field, to be aware and to transcend the limitations of the medium and the tools. In his view, the work of an artist is not relevant if it does not push the boundaries of the medium and tools. He expresses this by saying:

> *If you do not, as a digital artists, stay constantly aware of the limitations and drawbacks of the methods that you are using, you're derelict. .... You are not hitting the touch stones you should be hitting as an artist. (Appendix IV: Transcription of Skype Interview with Joshua Goldberg)*

According to Goldberg, an artist should also be aware of the transience inherent in choosing a medium such as software. The work is reliant on underlying technologies which change rapidly. An example of this is a work created with software may not be readable, playable or visible in a relatively short time due to changes in supported file systems or computers that are able to process the work become obsolete. He expresses the core of this idea by saying

> *I think we all need to realise ... that art that we are making currently with computers is ephemeral and temporary. There's no law that says that jpgs need to be an understandable file format in 50 years. And it won't be you know, CDROMs disintegrate hard drives die. You're making art for the decade, you are not making art for the ages. (Appendix IV: Transcription of Skype Interview with Joshua Goldberg)*

. Goldberg sees this as more important for artists who use digital medium than other medium. So Goldberg's process requires a constant questioning and reflection on the limitations of the tools and the medium as well as an awareness of the changes that happen in technology. A process that would assist Goldberg would need to be adaptable and would need to include regular reflection and questioning of the medium and it's limitations to make sure that the making pushes the boundaries of the tools and medium. The Agile process includes phases of reflection and questioning which informs making. It also includes experimentation with what has been created. The reflection and experimentation phases can be used to help the artist to create relevant work by Goldbergs definition. The Agile value "Responding to change over following a plan" confirms support for an experimental project approach which would suit Golderg's process.

For Proske it is important to take part in the coding process. As he expresses it "if you want to make something that's really original then you ... have to do it yourself or at least be really close to that process" (Appendix V: Skype interview with Pierre Proske). He describes this activity as being "heavily engaged in the medium" (Appendix V: Skype interview with Pierre Proske). Since he has shifted his focus away from using software as a tool to engaging with code as a medium, he experiences the need to keep a balance between engagement in low-level technicalities of the medium and being aware of his artistic goals and his conceptual intentions. He explains: "The more lost you get in the complexity of the implementation of the project the easier it is to forget what it was you originally

wanted to do" (Appendix V: Skype interview with Pierre Proske). So Proske's process involves a balance between conceptual reflection and immersion in making.

Proske has experience of engineering processes as he has studied in this field. He thinks that if an artist creates work that becomes collectible, that is, it is required to be accessible over a longer period of time, it may need more rigorous or formal engineering processes such as testing. He explains:

> *People working with code want to ... consider themselves artist ... being able to create works that are collectible... if, ... the art market will continue and then in order to be recognised as artists, code work creators would have to become part of that, then I think the software produced for the general public will have to be more robust and that will definitely require more engineering process and more testing". (Appendix V: Skype interview with Pierre Proske)*

For Proske the process needs to be adaptable because the longevity of the work and the exposure of the work may need different levels of repeatability and testing. The process would then need to adapt depending on the project.

Pall Thayer's process involves separating the conceptual framework of his work from the creative act. He describes it as follows

> *I had been exploring [sic] abstraction from an approach that involved the artist creating a conceptual atmosphere and then separating himself from the creative act, allowing interactive elements to take over and create the actual work. (Appendix VII: Email interview with Pall Thayer)*

The code becomes, as he puts it "the creative conduit that presents his concepts" (Appendix VII: Email interview with Pall Thayer). For this strategy the process would involve a phase of conceptual investigation, creating of the software and then engaging with the created work to verify that it performs as creative conduit for the concepts. Thayer is aware that the environment and medium in which he creates the software is not the same as the environment and medium which the viewer experiences the work. Thayer creates software which will execute his work in the viewer environment. The conceptual abstraction of Thayer's work is crucial to his process. The meanings and references in his work become encoded in the software. The meaning can only be unravelled by reading the code. The conceptual encoding and the writing of the code are intertwined. The code pieces have to be tested, because running the code validates them as software works. In this artist's process there is less of a separation in the conceptualisation and the making of the work. There is a separation in time and space of the concepts source code and when the computer creates the artwork for the viewer. Even if the different aspects of the process are enmeshed in this artist's process, the different elements, reflection and abstraction, implementation and testing are discernible and form a part of how this artist works.

In the interviews there is evidence of reflection, implementation and testing as part of the processes described by the artists. The different aspects of the process are used to different ends by the different artists. The different aspects and justifications of the process are driven by different characteristic of the medium. For example the technology changes quickly and Goldberg regards the digital artwork as relevant if the artist is aware of these changes and responds by creating

work that challenges the limits of the tools. The artist can do this by reflecting and by testing what has been implemented. The artist can challenge the limits by being flexible with the process that is used, adjusting as new limitations are presented. Proske balances implementation with reflection to make sure that he achieves his conceptual goals. Proske also thinks the process should be flexible enough to accommodate different levels of rigour in testing. Thayer merges reflection and conceptualisation with implementation and testing.

5.6 The Agile process and the creative process

The creative process improves the chance of a novel creation by being emergent. The Agile process is regarded as emergent due to its ability to respond to change and due to its iterative nature. The Agile process proposes self organising teams which also support emergence because a novel way of working is possible since the teams are not restricted by external structures.

5.6.1 Creative process models and Agile process compared

The history of software development processes progressed from a linear sequence of defined steps, requirements analysis, design, implementation, testing to an iterative approach. Although the creative process does not describe the same process as the software development process, there are parallels. The creative process was modelled in a linear way describing four stages, preparation, incubation, illumination and verification. The requirements analysis phase has cognitive activities similar to preparation. Likewise design has similarities with incubation and testing has similarities with verification. Both creative process models and software development models shifted to an iterative process.

5.6.2 Iteration

Iteration is one of the basic techniques of the Agile software development process. It is the fundamental mechanism which allows the Agile process to respond to fluctuating requirements. All the phases of the development process, design, coding and testing, happen in each iteration. At the start of the iteration the requirement driven direction can change. In the case of iterative creative processes such as the Geneplore model as described by Finke, Ward and Smith and the model described by Runco and Chand, the iterations oscillate between what they call a generative/ideation and an exploratory/evaluative sub-process, combining both processes in a cyclic fashion. In the generative phase multiple solutions are generated and in the exploratory/evaluative phase the solutions are tested for limitations (Finke, Ward, and Smith 18; Runco and Chand 245).

Finke, Ward and Smith use the term exploratory in the sense of exploring and choosing appropriate solutions and not exploring and generating more solutions. Runco and Chand prefer the term evaluative for this phase. This verifying of solutions in the Geneplore model can map to both the testing phase in an Agile iteration but also the requirement adjustment which happens at the beginning of an iteration (Finke, Ward, and Smith 18; Runco and Chand 245).

The fact that the Agile process values running software and a demonstration at the end of each iteration also supports artist who choose an exploratory cognitive style because there is running software at this stage which the artist can experience. If the artist needs to make adjustments in a different direction, this is also the time when the requirements can be adjusted because the requirements set

the goals for the next iteration. Once the requirements are set at the beginning of an iteration, the project runs using a goal oriented approach. This means that the Agile process can accommodate both exploratory and goal oriented ways of working with it's iterative approach.

Agile methodologies which favour fixed iteration times separate the exploratory phase and the goal oriented phases clearly. There is a time when iteration demonstration and requirement changes are done and there is a time when chosen iteration tasks are done. In practice, digital art projects may find that the clear distinction between the two approaches may disappear.

5.6.3 Collaboration

When tools and processes that support creativity are discussed, collaboration support is a re-occurring recommendation (Fischer; Shneiderman). Collaboration is one of the key elements of the creative process. The barriers of collaboration are categorised by Fischer as (Fischer):

- spatial - collaborators are separated by space
- temporal - collaborators are separated by time zones
- conceptual - collaborators do not share common understanding
- technological - collaborators do not share similar domain orientated tools or software systems.

The values of the Agile process and the creative process recommendations are aligned in this respect because both value and support collaboration. The Agile value "Individuals and interactions over processes and tools" as well as the value "Customer collaboration over contract negotiation" show the importance of com-

munication and collaboration as part of the basic value system on which all Agile processes are based (Beck et al.). Communication and collaboration are fostered in regular feedback sessions with all stakeholders of a project. Practical Agile process suggestions include daily face to face meetings and co-located teams. Both are mechanisms to break down spatial, temporal and conceptual barriers to collaboration. Collaboration is a priority for both the creative process and the Agile process and is ingrained in the values and principles on which all Agile processes are built. So for team management and communication the Agile process has tools to improve collaboration.

Co-located teams are sometimes not possible. Open-source communities can be investigated to see how teams use mediated communication to improve collaboration. The openframeworks community is an example of an online community where many of the participants never meet face to face. The communication happens on the forum, on a twitter feed and email news letter. In addition the communication and collaboration happens through the code which is hosted at GitHub. GitHub is a source code control service which gives free hosting to projects which are open source ("GitHub"). The open-source framework provides a software framework which can be re-used and re-mixed freely. All changes can be viewed on GitHub. In addition anybody can take a copy of the software do modifications and request that the changes be pulled into the main stream. So the source code control system functions as a self documenting framework.

The creative process can be described as "experimenting and iterating and throwing stuff out constantly, borrowing from other works to various degrees - from spiritual inspiration to borderline plagiarism - all in a churning creative pro-

cess to make something that's both novel and accessible" ("Rhizome | Do Artists"). Even though re-using and re-mixing as strategy pre-dates software and digital artworks, the digital nature and the mutability of software and digitised media artefacts facilitate re-use and remixing. Of course re-using someone's code or material does not mean there is collaboration among the parties involved but the facility with which material and code can be re-used and re-mixed supports collaboration. Various authors on creativity process stress the importance of collaboration (Buss; Candy and Edmonds; Amabile 84).

Smith, Mould and Daley describes what they call the infusion principle which states that software for artwork provide libraries which can be re-used and refined (82). This is based on ideas that creative processes need cultural input. As one participant in an online discussion in the openframeworks forum explains: "It's very difficult to juggle the myopic, complex intricacies of low-level programming with the broad sweeping idealistic ambitions of a playful, creative mind. One solution to this, has been to stand on the shoulders of giants and reuse other people's code in abstracted form. The open source movement has greatly assisted this endeavour" ("Has being an artist changed the way you code or vice versa - openFrameworks forum").

People also code add-ons to the basic framework which support anything from Kinect sensor support to particle systems (George, Borenstein, and Hughes). Using this internet based source code control system and opening the source code up to everyone, overcomes the technological barrier of collaboration. It provides a framework to jump-start a project but it also provides a way to share code with other users of the framework. Even if an add-on isn't directly usable, it provides a

coded documentation of how problems were solved which can help someone solve a problem indirectly. Collaborating with this community alleviates Proske's problem where he sees that the complexity of projects can cause him to focus on implementation rather than on his conceptual goals. By sharing implementation layers with the community, he can re-use code and spend more time exploring the conceptual frameworks of the work he is creating (Appendix V: Skype interview with Pierre Proske).

5.6.4 Flexible, simple and adaptive

The creative process recommendations, summarised in chapter 2 section 2.5, suggest flexible tools which allow for experimentation but are also simple to use. From the Agile value "Responding to change over following a plan" we see that flexibility and adaptability is built into the Agile value system. Abrahamsson et al. Highlighted characteristics which all Agile process have in common. The two relevant characteristics are "doing the simplest thing possible" and "being continuously adaptable" (Abrahamsson et al. 93). So if the Agile development process is seen as a tool, in its intention at least it supports these recommendations for tools that support the creative process. In practice using self-organising teams, and working in iterations where adjustments can happen provide useful tools for a project to remain adaptive and flexible. Performing continuous testing, if the exploratory approach to testing is followed, can provide feedback about the project and can be used as a mechanism to encourage experimentation. Responding to feedback is a way to maintain adaptability. So since the Agile process promotes regular feedback and has a value that states it is important to respond to change,

the Agile process has a mechanism which supports adaptability. If this feedback response mechanism is used in an iterative way, the Agile process is able to support adaptability for creative processes.

5.7 How does this process compare with the Agile software development process?

From the interview discussion in section 5.5 we see that it is important for artists to focus on the conceptual aspects of their work and that a proposed process or tool should support this and not distract from their creative practice. Since the interviews focussed largely on software as medium and conceptual discussions this highlights again how important these aspects are to the artists. These issues are not directly supported by the Agile process nor is the Agile process intended for this. What can be learnt from this then is that it is important for a process, such as Agile, that it does not hinder the creative process.

Comparing the creative process recommendations with the Agile process showed multiple areas where the recommendations of both are aligned. The Agile process model has similarities and overlaps with cognitive creative process models. The Agile process supports iteration which is a key element of the creative process. Both the Agile process and the creative process recommendations put a premium on collaboration. The Agile iteration, feedback adjust mechanisms, which is supported by the values on which Agile is based, provide a flexible and adaptable solution. Both Agile process and creative process recommendations propose simple solutions.

Even though detail development process information was not obtained from the interviewees, the interviews highlighted the priorities of these artists.

However since key elements of the creative process recommendations and the Agile process are aligned, the Agile process may be used in a way that does not hamper the creative process.

5.8 Engineering process and software artists

Engineering software development processes, of which the Agile processes are a subset, can support the creative process in more ways than one. The comparison of the Agile process elements with both the creative process and the replies from the interviews have highlighted some areas which may be useful to artists. In addition to Agile process suggestions there are also general engineering process tools which may be used to support artists. This section answers the question: How can conclusions made from a comparison between the Agile process and the discussions held with practising software artists shed light on the areas where the Agile process can assist artists and areas which might be avoided?

5.8.1 Agile process elements that support software artists

The Agile process elements that were identified as useful to artists are:

- emergence,
- iteration
- collaboration,
- support for different approaches
- flexible, simple, adaptive approach.

Practical Agile process recommendations, build on these elements and can provide support for digital art projects. The basic mechanism taken from the Agile process, which a digital art project can use, is use an iterative development cycle

which coincides with a creative process iteration. This means that the software development oscillates between a divergent mode where requirements are explored, an implementation phase which tries to build something that can be demonstrated and a convergent mode where the results are evaluated and the requirements are adjusted. This iterative process should contain all parts of the development process, design, and testing. This means the testing is continuous and the testing becomes part of the mode where results are evaluated. The iteration with feedback and adjustment allows the process to be emergent and exploratory but still provides goals per iteration to ensure that projects maintain momentum. To support collaboration during this process, regular feedback with all stakeholders, especially at the end of an iteration before a new iteration starts is encouraged.

5.8.2 Engineering process elements that support software artists

There are engineering process activities which are not part of Agile which can also support software artists. The two activities highlighted in this study are testing and the use of source code control systems.

5.8.2.1 Testing as part of creative process

Testing can be seen as a formal step from a software development point of view. Testing can also be seen as a mechanism to find failures. This approach to testing may not be appropriate for software art projects and the creative process. This is confirmed by Proske who explains that the amount of testing is dependant on the longevity and exposure of an art work (Appendix V: Skype interview with Pierre Proske). Artists do perform testing in the form of investigating and exploring a

system. If the emphasis of testing changes to become an activity which is part of the creative process, it can be a useful tool for artists. The exploratory testing approach as described by Bach and Kanes can be used in such a way. Artists may be performing more tests than they are aware of because playing with a system to see how it behaves can also be seen as a kind of test. In addition playing with a system to see if it does what you want is a test which has validation aspects as well. So testing whether explicit or implicit can provide input for the generative/ideation phase and the exploratory/evaluative phase of the creative process. In addition testing can be approached from both a goal oriented or experimental direction. Whether formal or informal, test results can be recorded and become part of the documentation of the making process of an artwork.

5.8.2.2 Source code control as part of the creative process

From earlier discussion we see that software is a transient medium. Software is mutable since it can change while it is running. As an analogy the software process is also mutable because a small change can create a new version of the source code that has a different result, potentially create a new version of the artwork. This can be done without destroying the original work.

Many tools have been developed to support managing multiple versions. These tools, referred to as configuration management tools, include source code control tools. Source code control tools excel at keeping a history of changes and allowing the user to recreate an exact version. The use of these tools are seen as a way to control software versions and a way to ensure consistency and reliability from an engineering point of view. However, in the context of a creative process,

this management allows the artist to capture multiple solutions and be able to switch between different solutions without losing any work. So a software development recommendation which supports control allows for flexible experimentation, something which artist value and which is a core aspect of the creative process.

So source code control supports multiple elements of the creative process and can be useful to artists. Captures code and coding history and this can be used as a backup for the situations where an artist does not want the code to be transient. It provides a way to explore multiple solutions by allowing the artist to create a different version without destroying the base version of the code. It gives the artist a way to jump between multiple versions of a project. In this way the transience of the medium can be controlled by source code control. Source code control can be used as an automatic documentation of the creative coding process. It can also be used to improve collaboration and to facilitate re-use of libraries.

5.8.3 Engineering process elements to avoid

Some parts of the engineering processes proposed are not suitable for art projects. Some parts are suitable but the focus of an activity needs to change for it to be useful. Since the Agile process is adaptive such modifications can be accommodated. Creative process vary and development processes are just tools to support the creative process. This means whatever adjustments an artist needs to make or whatever parts of a process an artist needs to discard are choices an artist makes as part of the creative process.

Some Agile methodologies propose a fixed iteration cycle. Also requirements do not change until the start of the next iteration (Abrahamsson et al. 30). Applying these rules to digital art projects may stifle the natural oscillation of the creative process of an artist. The Agile process iterations should not be applied to projects if it does not enhance the creative process iteration process.

Automated tests may not be appropriate because they may not provide enough benefit for the effort of developing them for artworks that have a large visual component and for artworks that have random or ambiguous elements. Art projects typically do not have contractual obligations that would merit automated tests.

Similarly scripted testing may be less useful than exploratory testing since the effort to develop them is not justified as it may not support the creative process. Testing is an example where the activity becomes useful if the focus and intention changes. So if testing is used to support the goals of the artist it is useful.

5.9 Creative process elements support coders

Being aware of different cognitive style approaches may assist a team to find a solution which would otherwise not be reached. The exploratory approach could be particularly useful for projects where the goal is not to find the most optimal outcome within the capabilities of the tools and medium but rather where the goal is to explore the limits of the tools and medium. Recommendations for tools that support creativity, stress that it should be easy to make multiple solutions and should be flexible so that different work-flows can be accommodated. Tools

such as source code control systems already support creating multiple versions of code easily.

This focus on flexibility and exploration as goal of a project will help software development process to respond quickly to change. Responding to change and accepting ambiguity as an essential element of the requirement definition process may assist the software development project to focus on the key Agile value of responding to change over following a plan. Testing is a key element of development and different testing phases have different goals. If it is not possible to automate repetitive tests as proposed by some Agile processes, an exploratory and investigative approach can be applied to tests which are not automated. This shift in focus can help to make sure that tests are done continuously during the development iterations. Being aware of multiple approaches to creativity provides a solution to the conceptual barrier to collaboration and can enhance team communication.

5.10 Process is not the whole picture

Recommendations that support the making process, whether creative or software development, are not intended to address the conceptual aspects of software art. These conceptual aspects of software art are discussed in the relationship between software art and conceptual art, the use of software as muse and as metaphor. The questions that were posed to the interviewees, asked about process and medium but they were open-ended by intention so that an idea of the personal priorities of the artists could be obtained. Only one of the artists discussed software development process even though they all had experience in

creating code. The one artist who discussed the engineering processes did so after explicit questions regarding this topic were posed. In addition I received no feed back on software development questions from the online questionnaires which were sent to a group of artists. This does not mean that artists do not apply any software development process it just means that if they did use any specific processes, points relating to the concepts and the medium took higher priority when they answered the questions.

Taking heed of development process suggestions, which have proven to be worthwhile in the software development industry, may well provide insights for artists to improve their work-flow, but choosing to create a software artwork has conceptual implications which affect the choices that the artist makes. These conceptual considerations should override development process suggestions. On the other hand conceptual considerations are not important for engineering projects which have to provide a specific solution with time and resource limitations. So creativity process suggestions may expand the possible solutions produced by an engineering team but conceptual choices should not jeopardise the efficiency of the team or undermine the engineering goals of the project.

6 Practical work

6.1 Installation description

Our mediated communication with each other and our interaction with the objects and systems in our environment pass through layers of software. From time to time we become aware of the software layers. This usually happens when they stop working but for the most part these software layers are so integrated into how we use computers and digital media that we don't think about them or how they are made. Furthermore the structure and workings of these fast running software systems are hidden from us by design (Bunt, "Risking code" 10).

Software is constructed so that details and inner workings are hidden by software development techniques such as encapsulation. This technique allows programmers to use modules of code without knowing how the logic is implemented. Details of lower level functionality are also hidden by using a layering system where different software systems provide different layers of functionality. Systems that use the lower layers do not need to know how they work. What is presented to the end-user is an interface into complex layers of code spanning multiple machines which cannot be understood by one person at any one time. Hayles expands on the view that all computer mediated communication passes through layers of computer code which is inaccessible to most people. She draws the analogy that the relation of software to language becomes similar to the relation of the unconscious to the conscious (Hayles, "Traumas of Code" 136-137).

Furthermore computers execute the commands that make up these complex layers, at millions of instructions per second, so quickly that it is impossible for a

human to perceive. The way software runs makes it hidden to most people (Wright 61). Wright explains this by saying that once software is running "it takes place on such a fine temporal and symbolic scale and across such a vast range of quantities of data that it has an intrinsically different materiality than that with which we are able to deal with unaided" (79). According to Bunt, one of the dilemmas which software art faces is the dilemma of visibility. When software is running it runs so quickly that the working is not seen, however software art attempts to make the medium (software) more visible ("Risking code " 13). Bunt suggest that the artist engage with the nature of the code to hide details and use this as a "field of poetic potential" ("Risking code " 6). The practical work accompanying this thesis attempts to make aspects of the software development process and aspects of the way software runs visible to the viewer player. It also attempts to visualise the differences in the way machines represent ideas and the way humans represent ideas. It utilises player interaction to trigger different visual and audio elements which are interpretations of either human or machine activities as well as software development activities.

The intention of the practical work then is to play with the visibility of the underlying software, effectively drawing it to the forefront and allowing it to recede depending on the player presence. The software installation pieces in this body of practical work, titled *nullPointerException*, hide and reveal interpretations of aspects of software code through interaction. The installation works use the Microsoft Xbox Kinect as its primary sensing device for this interaction.

The Microsoft Xbox Kinect is the interaction sensor usually used in conjunction with a Microsoft Xbox gaming device. It is originally intended as an interface to a

gaming platform and used for entertainment purposes. It is capable of detecting multiple people in a room and detecting how far they are away from the Kinect. It can also detect bodies and limbs as well has hand gestures. It uses an infra-red grid of dots and two infra-red cameras to detect the players. In this installation the interaction feedback received from the Kinect is passed to the software artworks, which then adjust the visuals in the system accordingly. Interaction can only happen in the range that the Kinect sensor can detect.

From an interaction point of view the player changes elements simply by being there. So there is direct interaction with the visuals. The interaction controls the hiding and revealing of the different elements of the work. I have therefore intentionally designed 'safe spaces' as part of the engagement for this body of work: spaces where players are not detected, spaces where players can view the work without interacting with it. Players can also observe the way the work changes when someone else enters the space, thus setting up an indirect interaction among players. The installation of the work was done in such a way that another work is partially visible from the main focus area of one work. This creates an interaction among works through the experience of the player. This body of work consist of 4 individual interactive pieces which relate to each other to form a coherent body of work.

A *nullPointerException* is an error message reported by the computer when the running code tries to reference something that does not exist. It is an example of a way that the machine interacts with the humans who program it. To check that the code makes sense to the computer it is compiled (in the case of c++) and then run. The software is tested to check that the behaviour is as the programmer in-

tended. During run time if a condition occurs that the programmer did not envisage the software may crash and report an error message, such as *nullPointerException*. These works refer to the fascia/interfaces between human ideas and machine execution.

The titles of the individual works are contractions of truisms which programmers sometimes use as guidelines to improve their skills. The titles also highlight different aspects of software creation: the coding itself, the software architectural structures, the creation of multiple versions of the source code and the effect of algorithms and logic on the behaviour of the software. There are four works in this practical presentation: *commentCompile*, *interfaceInstead*, *commitOften* and *initBefore*. Video documentation for each of these works can be accessed on the accompanying disk or online.

- *commentCompile* in the video titled nullPointerException-commentCompile.mp4 and (Grotepass, "1 nullpointerexception-commentcompile")
- *commitOften* in the video titled nullPointerException-commitOften.mp4 and (Grotepass, "2 nullpointerexception-commitoften")
- *initBefore* in the video titled nullPointerException-initBefore.mp4 and (Grotepass, "3 nullpointerexception-initbefore")
- *interfaceInstead* in the video titled nullPointerException-interfaceInstead.mp4 and (Grotepass, "4 nullpointerexception-interfaceinstead")

6.1.1 commentCompile

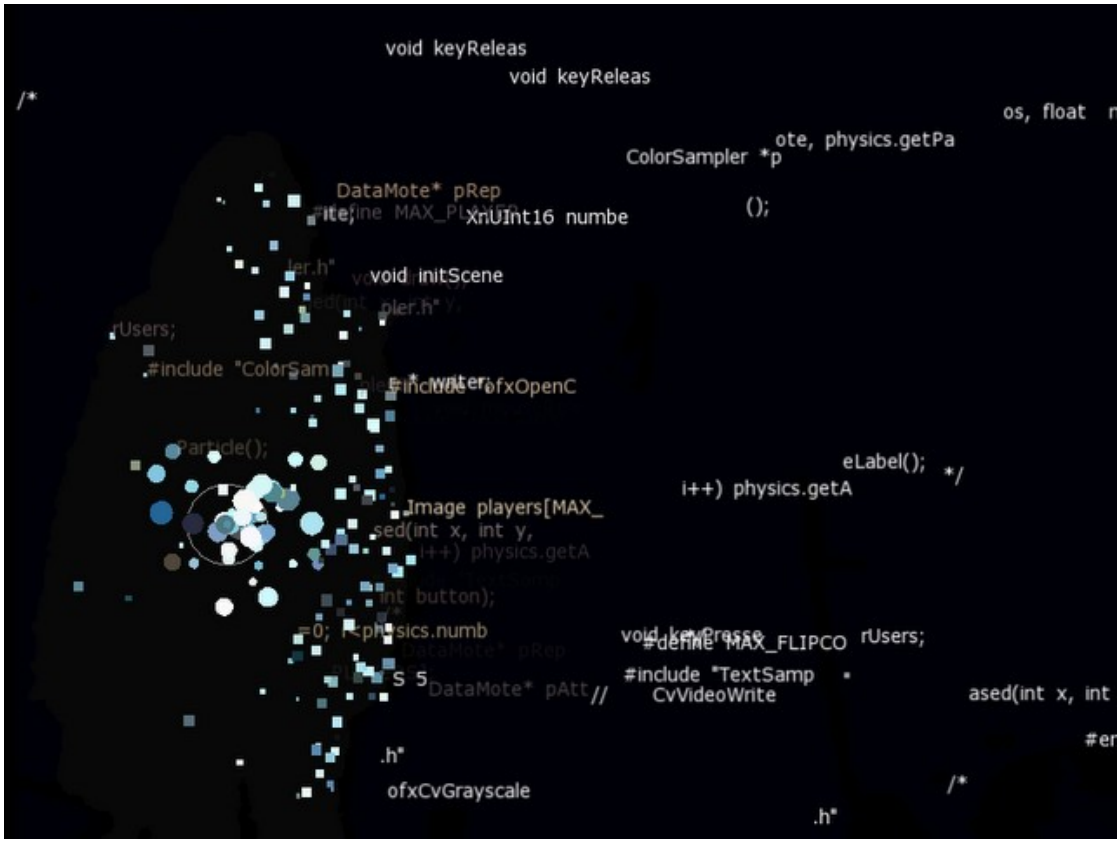


Fig. 21: commentCompile screenshot

Source: screenshot of commentCompile software artwork

This work visualises the process of putting together arbitrary code syntax constructions into a coherent machine executable unit; swarms of elements which coalesce into a body of code that has behaviour. The code contains no personal elements of the programmer but the results of constructed software contain idiosyncrasies and choices made by each programmer. So in this work the code snippets are attracted to different players depending on their spatial position and will follow a particular player tracing their movement. See fig. 21. An extract of the DataMote class which is displayed as part of this work is shown in Appendix II: Extract of DataMote particle class. From the code extract choices that the program-

mer made and text that the programmer chose to include in the comments can be seen.

Code captures the behaviour that the programmer wants the computer to perform. The code has to be constructed in such a way that it can be executed or run by the machine. A human can make sense of some language elements even if parts of it is scrambled but a computer can not understand a program at all if the sequence is changed.

A computer representation of an image is recognisable to a human if the pixels are presented in order. If the pixels are scrambled the image becomes unrecognisable to the viewer even though the computer has enough information to recreate the image. Both the code view and the pixel view contain information only accessible by the computer system or the human viewer. In this work the visible and invisible juxtapose each other from both the human and the machine perspective. Some of the information encoded in the structure and visuals of this piece can only be decoded by a human and some can only be decoded by a machine effectively rendering other parts of the information invisible to the machine/human.

The hiding and revealing of visual elements that cannot be deciphered by either the computer or a human highlights the difference in the way humans and machines perceive and make sense of data. People require pixels in sequence to be able to recognise an image. Computers require code in sequence to be able to interpret and execute the code.

6.1.2 commitOften

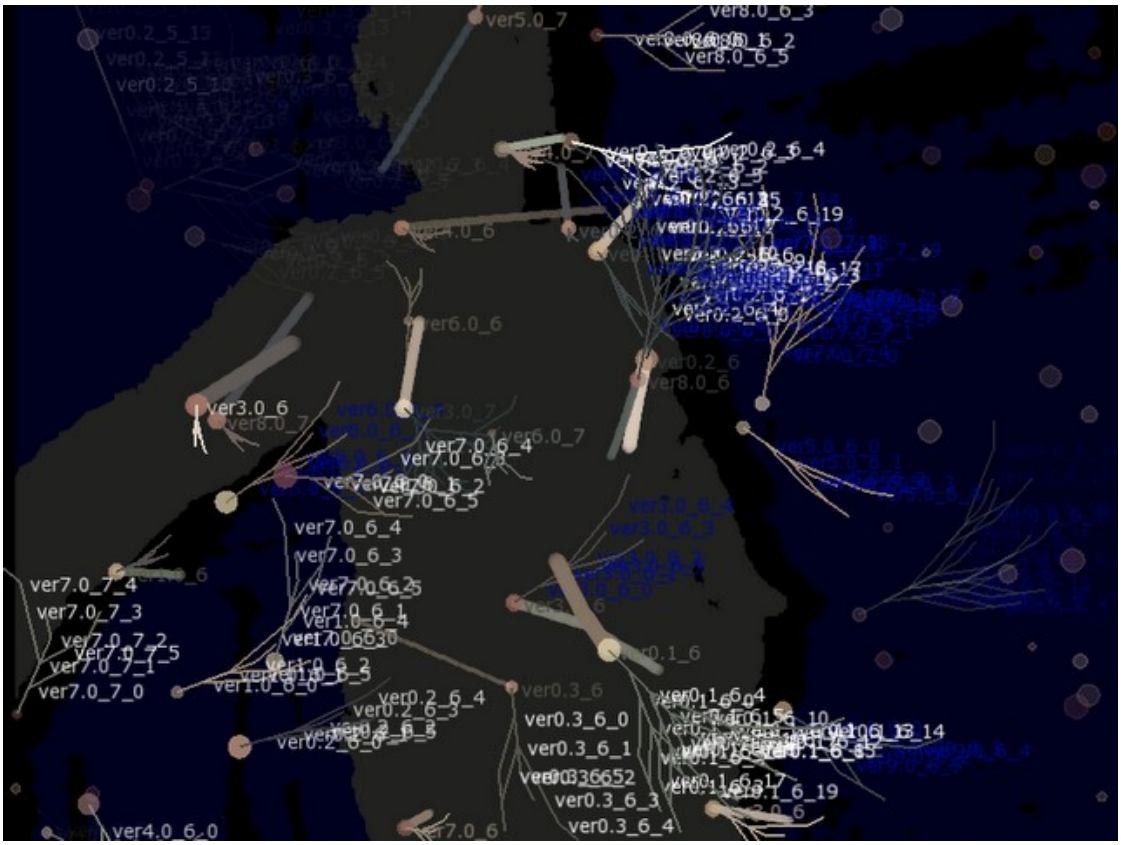


Fig. 22: commitOften screenshot

Source: screenshot of commitOften software artwork

Software is being embedded in devices around us. Software elements and hardware go through multiple iterations. As soon as a human changes any part of a software stream it branches into a new version. This branching and merging process of software around us is hidden, as only a particular snapshot is released to consumers to be used. This branching and merging process can be observed when one looks at open source software development as the code and development process is not hidden.

The multiple versions of code are kept in a source code control system. In the software development process the process of coding is often followed by a cap-

turing of the changing into the source code control system. This action is called a 'commit' in some systems.

In this work each particle contains a history which is invisible until the player interacts with it. Interacting with the particles trigger branches and version activity. See fig. 22. This work visualises how software streams do not change until developers interact with them, causing the streams to fork and merge into multiple versions. The multiple versions refer to Manovich's principle of variability on a conceptual level (Language of New Media 55).

The interim versions are tracked but they are transient because they are superseded by new versions as soon as something changes. So in the work the branched version numbers fade away when the change agent, the player, moves away. This transience is also shown by the brief, illusive displays of the player silhouette which disappears. Each circle contains a history of its movement and its version numbers, which become visible on interaction, when there is no interaction the particles revert to an almost static state, echoing the state of archived software streams.

6.1.3 initBefore

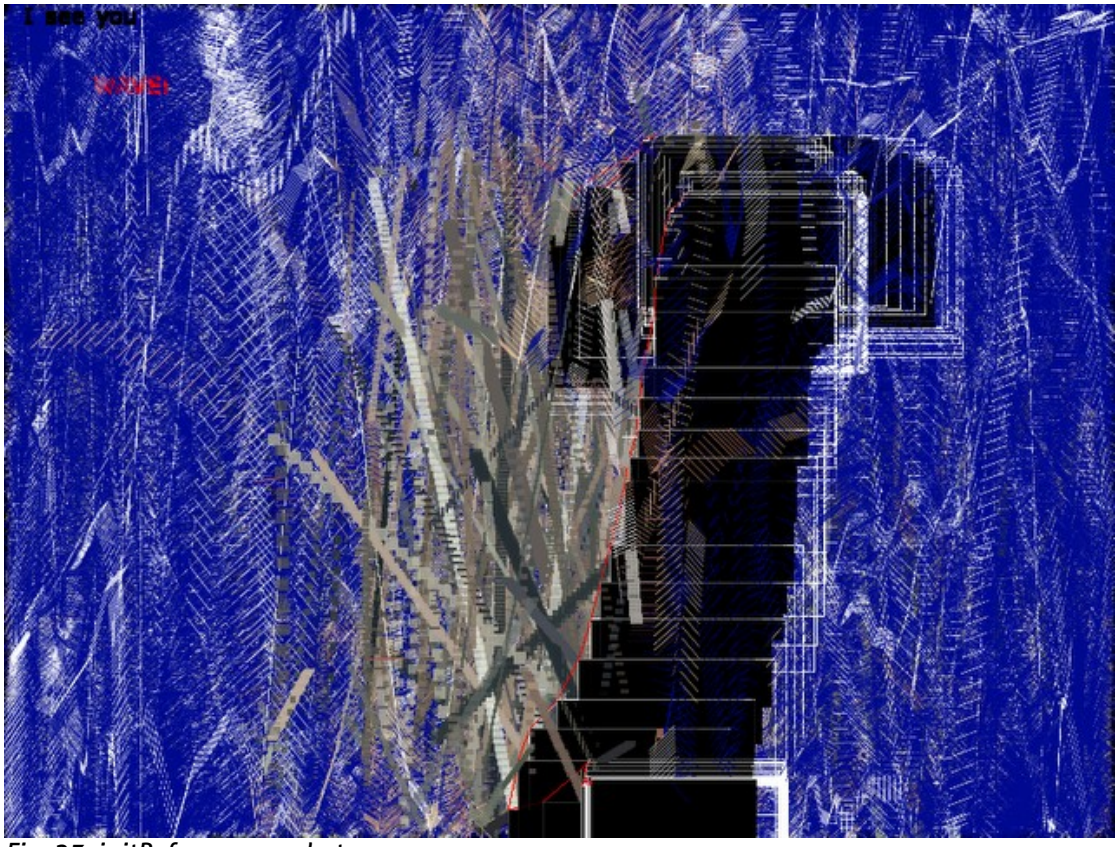


Fig. 23: initBefore screenshot

Source: screenshot of initBefore software artwork

Computer memory, volatile and non-volatile, record bits: zeros and ones. Software writes and rewrites in memory. The values writing in memory do not make sense unless they are interpreted in the right way. Reading out of memory at a location but shifting the data by one bit makes it completely unintelligible. So the blue and white hatch marks, the two colours only, fill the field of vision in a way similar to computer memory being written and overwritten. The exact placement of each line is controlled by an algorithm much like computer memory and disks are filled with ones and zeros by algorithms. Just looking at a filled disk does not help to understand the data.

Repetitive algorithmic filling in of the space that corresponds to the player in the room reveals the figure which becomes recognisable to the player. The edges of the silhouette are ragged because humans do not need clear edges to recognise something. See fig. 23.

When a player is detected by the software a waving hand prompts the player to mimic the gesture. A specific gesture alerts the machine that the player wants to interact with the work on another level. Once the gesture is recognised, the piece tracks the player's hand by clearing a black box over the hand position. The partial clearing of the visual field mimics the way a programmer would initialise areas of memory to known values. The block that tracks the hand has another purpose in that it reveals the individual hatch marks by obscuring and overwriting the historical layers of marks underneath. In this way it acts a lens to investigate the hatching algorithm. So by hiding the history of marks that occurred before, the working of the algorithm is revealed.

This work deals with the behaviour change over time using algorithms that reveal and obscure the view of the space over time. If the player stays in one position for a while, a silhouette emerges as a positive shape pushing forward and shifting the blue/white field into the background. The individual hatch marks happen quickly, much like machine instructions that can't be observed by a human. Interaction with the work reveals the process but alters the work. This is similar to a software debugging session which necessarily alters the way the code runs to allow the human to observe the process.

6.1.4 interfaceInstead



Fig. 24: interfaceInstead screenshot

Source: screenshot of interfaceInstead software artwork

This work has two clear modes of presenting itself. The mode where the work is aware of the player and the mode where it is not. It uses sound and visuals to distinguish the two modes. The two modes form binary opposites of each other. Fig. 24 shows the work just after the work has become aware of the player. It shows fragments of the machine world-view mode being covered by the human world-view mode.

In the mode where the work detects the player, elements that indicate conceptualising and planning made in a non-computer environment are presented: scribbled words, small gestures and chalk erase marks. The individual images were scanned from the physical planning documentation for this work. Mark,

movement and hatching was investigated and the movement of a pen tip was explored to understand what algorithm would be needed to recreate them in a digital environment. The scanned gestures scribbles are the seed of the movement that each element on the screen performs. So the image performs little gestures as if each particle is a pen point. This phase continuously layers and obscures underlying images. The scans chosen for this phase are intended to highlight the human aspect of the software planning and creating process. A sigh, yawn or cough often induces people within hearing distance to respond with a similar sound in an empathetic response. So the sound prompts the players to react to the work with a sound. The sounds visuals and colours are familiar.

Initially, but very briefly the previous phase is visible, but it is quickly overwritten and obscured by layers of images. In the phase where no players are detected, no history is kept. Each frame is redrawn. Visual elements that echo the software class structure and hierarchy moves around the screen in a linear way. Operating system error messages, that change quickly, are interspersed among the geometric shapes. The error messages are changed so quickly that it is impossible to read them. This makes the player aware of the quickly change elements of running computer software that are invisible to humans because they are changing and running very quickly.

The partition that separates the player from the work bears similarity to a bird-watching hide screen, communicating that the player can observe without being detected by the piece. The sounds chosen for this phase of the work were switches being flicked on and off and the monotonous beeps of a computer starting up. It underlines the machine nature of this phase of the work.

Machine nature and the human familiar nature of the two modes contrast with each other on multiple levels. The human recognisable mode has history, overlays images over each other, uses organic structures, uses handwritten words, uses many subtle colours which are close to each other, uses transparency, uses words and letters of varying length, uses small fidgety random movements and uses sounds which relate to humans. The machine mode renders each frame algorithmically from scratch, does not use overlays and transparency, uses geometric structures, uses fixed width machine rendered fonts, uses only three colours, black (R:0, G:0, B:0), white (R:255 G:255 B:255) and blue (R:10 G:0 B:145), uses vertical and oblique geometrically derived movement and uses sounds relating to switches and computer boot sequences.

In the process of designing all the works, but especially *interfaceInstead,* I first explored/tested how an element would move or how elements would layer by drawing the marks on paper. This was done so that I could understand the speed and velocity implications of a group of pixels drawing on the screen by physically making marks on paper. See fig. 26. The way visuals layer and blur was explored by layering strips of paper and working with chalk and crayon. Examples of this process can be seen in fig. 25 and fig. 27. This knowledge was then captured in algorithms and scans within the software and data of the work. Drawing the marks one by one is different to the way the machine draws. The machine also draws the marks one by one but because the machine rendering is very fast, it seems as if the multiple marks are drawn simultaneously. These test marks were scanned and form part of the visuals presented in *interfaceInstead*. Digitising the test marks converts them into a content element which can be manipulated al-

gorithmically. This is a conversion from the way a human experiences a visual element, such as a chalk mark on a piece of paper, to a trans-coded form that is accessible to the software. The software again renders the digital format of the image into a format that can be projected and perceived by a human in physical space.

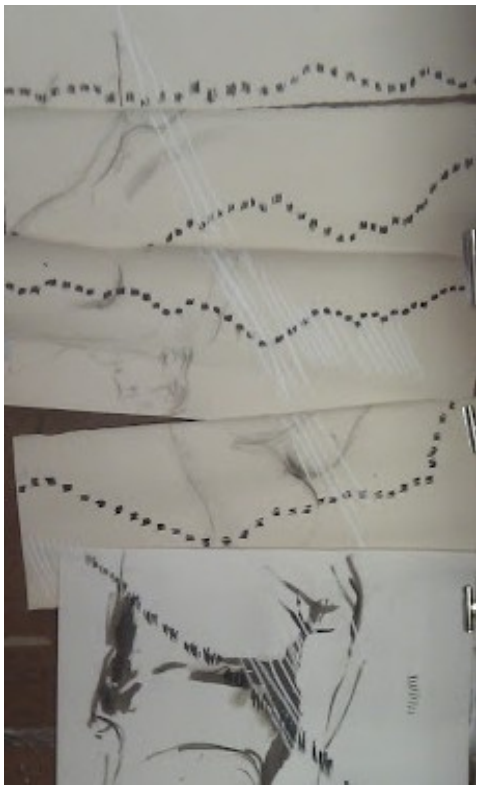


Fig. 25: Mark making investigation
Source: Photograph M Grotepass 2011



Fig. 26: Mark making investigation
Source: workbook scan M Grotepass 2011



Fig. 27: Mark making and layering investigation

Source: photograph M Grotepass 2011

6.2 Artistic choices

6.2.1 Spaces and screens



Fig. 28: Floor-plan showing installation spaces and screen placement

The works *commentCompile, commitOften* and *initBefore* were placed so that they react immediately as the player walks into the room. The initial planning of the works in the space can be seen in fig. 28. These three works refer to coding activities and source code control activities, the one activity leads into the other, coding practice followed by debugging. So the movement of the player between these three pieces mirror the typical coding cycle of coding, debugging and committing and coding again.

All the works were installed in a way that they could be viewed without the sensor seeing the player. This allowed the player to observe without changing anything. This passive position of observer is not the obvious and first encountered position but rather one that has to be searched out by the player. Projection on translucent screens allowed one player to interact or perform and other players to observe the pieces from a passive vantage point. This arrangement underlines the hiding-revealing opposites explored in these works.

The definition of the word screen has two meanings: it can refer to the area on which images are presented or it can refer to a construct that hides images. The arrangement of *commentCompile*, *commitOften* and *initBefore*, creates a space where the player is being screened from the sensor by the surface on which the image is projected. From one side the player is seen by the sensor and the visual elements created by the works remind the player that she is in the image; somewhat like a distorting mirror. Viewing yourself in silhouettes that appear briefly or silhouettes which are built up over time, makes the screen be a mirror of sorts presenting the player immersed in the work. Stepping behind the screen then allows the player to step behind the "mirror" or to step behind the "computer screen" into the world on the other side and to view it from a different vantage point in mirror image.

The choice of a projection on a screen with a horizontal aspect ratio refers to the computer screens on which code is usually created and inspected. The screens were suspended making the attachment almost invisible in the dark so that the effect of hovering computer screens was created. Suspending the screens away from the walls created a different space on both sides of the

screen. This allowed the player to step behind the screen reminding the player that there is a view from "inside" the machine.

In the works *commentCompile*, *commitOften* and *initBefore* the computer installation and projection equipment is separated from the player and sensor by a screen on which both the computer renders and the image of the player is made visible. The player is not visible to the software system if it isn't detected by the sensor and a rendering of the software execution is not visible to the player without the screen layer. The choice of flimsy translucent material for the screen with almost invisible support is intended to create the sense of a membrane. It is the membrane construction that separates the two worlds that also makes the one world visible to the other.

The three works in the main room *commentCompile*, *commitOften* and *initBefore* are separate from the fourth work because they relate to the implementation process of writing software. Writing code, committing to source code control, making a new version and testing the software algorithm is a cyclic activity . So these three works were placed in one space so that the viewer can move freely among them and be aware of all three, effectively cycling through the process by moving in the space. The software development activities that these three works refer to, all form part of the implementation phase of the software development cycle. This also mirrors the cylcic nature of the creation process as it manifests in software development. New code is written, recorded and then run to see if it provides a solution. The solution is then verified and the code adjusted and refined and the cycle continues.

Before implementation happens, the programmer has to design the software part to be implemented. The software development cycle can return to the design phase but the design phase is a different cognitive activity to the implementation phases. The design phase involves creating a mental model of the software structures. When models of software structures are created a certain aspect of the software is isolated and designed independently from other aspects. So a model provides a partial view. For example the software may be designed first from a static point of view showing relationships to the different sub-units. In object-oriented design this would be the class-diagram view of the software or the object instantiation view of the software. Then for example, in a separate exercise a model would be made of how an object behaves. For this one could use a state diagram. This segmented view of the software which one uses during design, informed the choice of screen and partition of the work *interfaceInstead*.

The work *interfaceInstead* does not make use of the translucent screen construction. This work has a constructed screen with a slot which provides a partial view of the work while screening the player from the sensor. The solid opaque screen is constructed to echo the shape of a bird watching hide or screen. This construction accentuates the player hiding from the sensor and allows the player to observe a different view of the work by looking through the slot. Once the player moves out from behind the hide screen, the work detects the player and shows a different view. This work has the projection screen against the wall because the design phase can happen without any execution happening on the machine level. So the machine execution has to be imagined by the designer and is

not directly involved with the process. For this reason the physical computer installation is on the same side as the viewer.

The hide screen construction also creates a physical spatial representation of an OR gate. An OR gate is an electronic device which will switch on if one or more signals are present and switch off if no signals are present. So *interfaceInstead* will present the human view if one or more humans are visible and the machine view if none are detected. The machine view is only visible if all players are looking through the slot and not standing in the sensor range. This allows the human presence in space to switch state in the artwork in a similar way to which electrical signals would change signals.

The the implementation phase and the design phase involve different cognitive activities. This is another reason why *commentCompile*, *commitOften* and *initBefore* form a unit and *interfaceInstead* is treated differently. The design phase requires the programmer to identify concepts and decide on software structures that have to be created to match her ideas. The concepts described by the human programmer have to be implementable, so the programmer must be aware of how the software constructs can be represented in the computer. This is also a phase that can happen completely in the mind of the programmer and not involve the computer at all. A design can be recorded with pencil and paper or live only in the mind of the programmer. For this reason *interfaceInstead* is the only work that uses scanned marks from my workbook. For this reason too, the mode where the human view is presented only uses scanned marks and eventually fills the entire screen with human marks. In this mode the sound supports sounds clips which are recognisable as human sounds.

The design is done to be implemented ultimately so the concepts have to be converted into a format that the machine understands. Thinking in a human way but always bearing in mind how these concepts will be translated into a machine representation accentuates the difference between the two modes for me. Since the design phase does not translate directly to the machine, it requires an implementation, there is no guarantee that the design is what is captured in the final code. In addition the design may not be complete, the designer, in an attempt to simplify aspects to an understandable level, may not have anticipated all behaviours possible. When the code is implemented and run, this disconnect manifests in unpredictable behaviour and error messages such as the error messages presented in the mode when the work does not detect a viewer in the room. So the Blue Screen of Death or the error messages is a communication from the machine that there is a disconnect in the way the programmer sees the software and the way the machine executes the software. This again refers to the title of the exhibition. The title is also an error message which the machine uses to report the same programmer machine disconnect.

In all four works the player can choose to be seen and engage, or to recede and observe and become hidden - invisible to the machine. In all four works software and machine constructs are hidden and invisible to the humans and are revealed when the player chooses to engage with the work.

6.2.2 Interaction strategies

All four works were created with multiple interaction mechanisms in mind. The initial aim was to make a visible change as soon as someone moved into the

sensed space. On this level just being in the space made the work react differently. This simple interaction was used to make the works more accessible on an initial experience so that the player does not feel intimidated and is willing to explore. In *commentCompile* presence made the code change to pixels. In *commitOften* presence made the code balls reveal branches and version numbers. In *initBefore* a waving hand appears and the hatching changes where a player is detected. In *interfaceInstead* the mode changes to something that is visibly much different to the previous mode.

If the player spent a little longer with each work, additional behaviour becomes visible. In *commentCompile* the code particles would flock and follow around the centre of mass of the person closest to the sensor. The particles would follow one person around until someone else steps closer to the sensor. Particles that have just passed over a player still retain code for a while but the code changes from white to an image sampled colour and eventually fades away. The pixels are initially rendered as squares and then transform to circles which become brighter as they come closer to the attraction point at the centre of mass of the closest player.

In *commitOften* the code branches change complexity if the player stays in the space. The branches that drift off the player slowly fade away. At unpredictable intervals the silhouette of the player is revealed briefly.

In *initBefore* the players in the scene are drawn more densely over time. If a player mimics the hand wave presented when a player is detected, the player can "draw" into the visual field with her hand and alter the image for a time.

In *interfaceInstead* the human mode oscillates between the drawing sub-mode and the erasing sub-mode. This becomes apparent if the player spends some time with the work. The drawing sub-mode has more gestures and words and has cough, throat-clearing, and sniffing sounds. The erasing sub-mode covers the previously drawn elements with transparent pale chalk marks and has yawning and sniffing sounds. The human sounds were chosen to provoke a similar sound response from the player. Humans tend to yawn, sniff or clear their throats as an empathetic response when they hear those sounds.

The software behaviour in all four pieces uses mechanisms which are ambiguous and not initially apparent. Some players guess at the algorithm which is used to control the visuals. The player engages with the software behaviour because the player is trying to figure out the rules by which it operate, thereby mimicking the behaviour in his or her mind. So the logic of the software is detected by the player using the observation of the visual behaviour of the particles. This level of interaction was observed at the opening night when people asked about the behaviour and tried to figure out if the way they understood the behaviour was the actual way it was coded. This layered approach to the interaction mechanisms was implemented to engage the player quickly in the beginning and to provide additional levels of experience if the player is prepared to spend a little longer with the work.

Trifonova, Jaccheri, and Bergaust integrate the different definitions of interactive categorisations in the literature (46). Their categorisation depends on three elements.

- how the content is generated: is it pre-define, is it generated by the software or does the player input content
- how is the interaction triggered: audience presence, audience action or environment
- how the content is influence by the audience or the surroundings: static or dynamic

From these categorisations the interaction mechanisms of the works are classified. All of the works use generated visuals. *interfaceInstead* combines generated visuals with pre-defined visuals in the form of scanned images. The works *commentCompile*, *commitOften* and *initiBefore* take content input from the player because the player in the space changes the generated visuals. None of the works are triggered by the environment. On the simple level all of the works are triggered by presence. The works *commentCompile*, *commitOften* and *initBefore* are triggered by human actions. In *commentCompile* and *commitOften* the action is the position relative to the sensor or the position relative to other people in the space. In *initBefore* the action is a specific hand gesture. All four works are dynamic because the content is influenced dynamically by the audience.

6.2.3 Colour

Colour choices were based on pixels sampled from an image. Two images were used to dictate colour choice in all four works. The first image is a web cam capture of the artist, fig. 30. The second image is a small piece from the Blue Screen of Death, which is blue (R:10 G:0 B:145) with white (R:255 G:255 B:255) letters,

fig. 29. Black (R:0 G:0 B:0) was chosen as a background colour as it is the binary opposite of white.

The two colour palettes represent the two viewpoints that these works speak about: a machine view and a human view. The limited palette of binary opposites, white and black, and blue represents the machine view. The muted subtle greys and pinks represent the human view as there is a direct connection to the human artist from the web cam image. The intentions of the coder/artist is captured in the code structure. The pixels of a visual web cam capture of the coder/artist is encoded in the colour values of each particle. So the face of the artist/coder becomes the palette that each particle uses in its drawing process.

The web-cam sensing devices samples the light coming through the lens and digitizes what is detected to form the colours of the web-cam image. So it is a conversion from light reflected off a human to a digital representation of the human. By contrast the colours chosen for the machine view are three specific values with exact red, green and blue values.

The source of the machine view colours is chosen as the Blue Screen of Death because this is one of the ways that the machine reports a mismatch in the human view and the machine view. The programmer tries to write software that does not fail and performs correctly but cannot foresee all possibilities and when the machine encounters such a situation it may crash and report a Blue Screen of Death. The machine is communicating a situation from which it cannot recover. The Blue Screen of Death is a conversion of a condition in the machine to text which is readable, albeit not always understandable, by a human. The specific error and memory area in which it occurred is reported as numbers. So the Blue

Screen of Death becomes the symbol of the difference and mismatch between human and machine representation of concepts. The machine is communicating that something in the software, which a human created, has caused a problem. The error report is presented in human language which could be understandable to a human but it is not because the software refers to elements such as addresses and codes which is not readily understandable. It is when an error occurs that the difference between human and computer view is revealed.

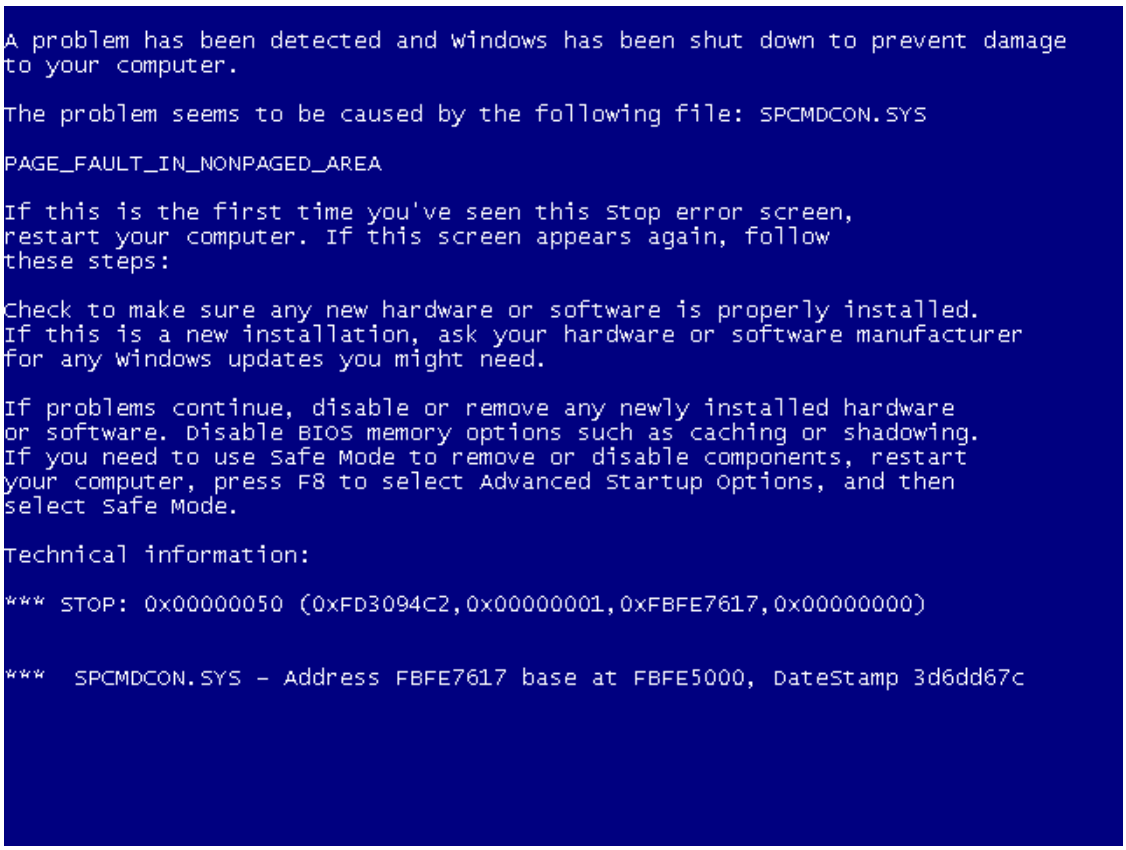


Fig. 29: Blue Screen of Death

Source: ("File:Windows XP Blue Screen of Death (PAGE FAULT IN NONPAGED AREA).svg - Wikipedia, the free encyclopedia")

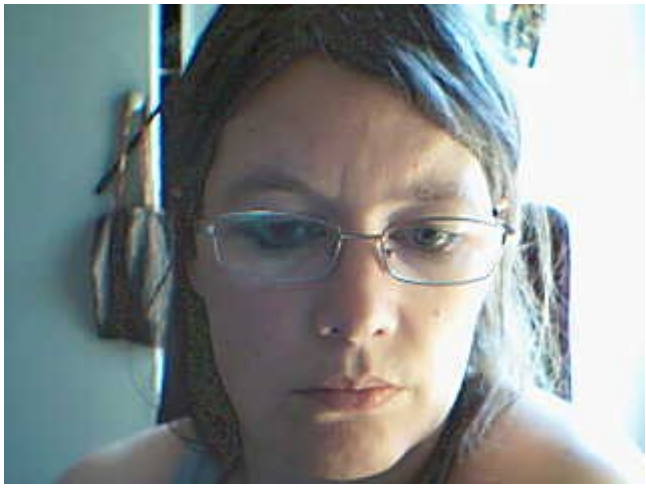


Fig. 30: Web-cam M Grotepass

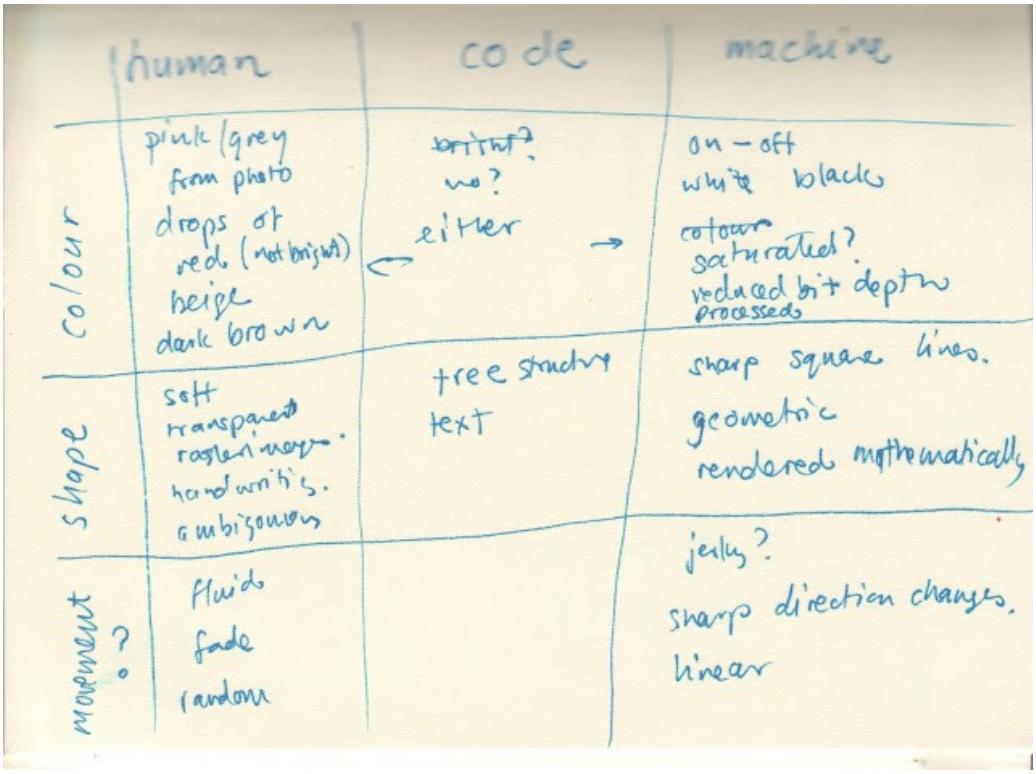


Fig. 31: Colour, shape and movement choices

Source: M Grotepass 2012

The colour treatment was repeated with variations in all four works. This was done as a visual reminder that the four works deal with concepts that relate to each other: different aspects of the development process and the juxtaposition of machine and human understanding of the code. The coherent colour treatment indicates that the four works are parts of a whole and the works together can be

read as one work with multiple parts. Fig. 31 is a scan from my workbook which documents the analysis of colour, shape and movement for the whole exhibition. Viewing the different works as part of a whole is necessary because the different works refer to sub-processes or phases of a software development and creative making process.

6.2.4 Font

Two fonts were used in the works: Verdana and Lucida Console. Verdana was chosen for *commentCompile* and *commitOften* because Verdana was created to improve readability on-screen and it is most often viewed in the context of a computer screen. It is an ubiquitous font, often avoided by designers for this very reason. For these works I wanted to communicate that the text refers to activities that happen in a code creation environment, in front of a computer screen. It is the font used by humans to write code which the machine must compile, interpret and execute.

In the case of Lucida Console. it is the font that is used by the error messages reported when the Windows Operating System crashes. See fig. 29. It is a fixed width font, that is, each letter in the alphabet takes a fixed amount of space and all letters have the same fixed width. It is used for code alignment where the strict lining up of code structures carry indentation and code structure meaning. e.g. statements of code that form part of a conditional code block will be indented more than code outside the conditional block. It is associated with low-level computer messages and code that is closer to the machine than words that are

formatted for human readability. Lucida Console is a font used by the machine to communicate an error condition the human must interpret.

In the work *interfaceInstead*, Lucida Console contrasts with the handwritten word fragments with variable width, size, opacity and spacing. This contrasts underlines the difference between the evolving of human concepts and the way those same concepts are represented when the code is constructed. The multi-layered ambiguous renderings of human writing are compared to the fixed width view of an error message.

6.3 Technical implementation choices

6.3.1 Tools and hardware

The Xbox Kinect was chosen as sensing device because it is accurate and gives suitable player presence information. It works in variable light situations, especially a dark environment which is suitable for projected images, because it uses infra-red. It is more reliable than web-cam or ultra sound sensing devices, it has open source drivers available.

Ubuntu as operating system was chosen because it is open source, there are Kinect drivers and the openframeworks platform is supported. The openframeworks development libraries and tools were chosen because it allows low-level access in C++, so if there are performance problems, the coder has access to the code. Equally important though it has a large community supporting it and it has many open source addons for the Kinect, for particle systems and xml variable reading.

The choice of Ubuntu, openframeworks and Kinect was done because it meant that the software for the driver, framework and platform is open. This allowed me to jump start the project with utilities and libraries which are free to use and change. Equally important as conceptual choice, is the possibility of sharing my code with the community. The code is available to the community in a public source code control repository (Grotepass, "maiatoday (maiatoday)"). The work does not stand in isolation. Segments of the source code were made visible, particularly in the work *commentCompile*, as part of the visuals of the work, but the entire development tree with all the history of all the changes is available for comment and investigation online. This includes the other projects I used to build parts of my projects and the changes I made to these projects. Cramer distinguishes between software art and software-based art by saying that software art "exposes its instructions or its codedness" (7). So the installation is the visual and experiential manifestation of the work but the code repository records a history of the source code and all elements which were re-mixed to achieve the result. This also allows new work by me or by other artists to grow from this collection of code by following software development process, the same software development process that was the starting point for this work. So the choice of software environment was as much a practical decision as it was based on a conceptual underpinning of connecting the work to the process to which it refers.

6.3.2 Development process

The development process was not strict, there was no team. Weekly and sometimes daily to-do lists provided a mechanism to keep the project on track. A work-

book was kept to record activities. The benefit of the workbook is that planning and to-do lists were recorded in the workbook. The workbook also allowed for ideas to be recorded and explored. In this way a goal oriented approach could be supported as well as an exploratory approach could be recorded. Looking at the workbook both approaches can be identified. See fig. 32 and fig. 33 for an example of goal oriented documentation and see fig. 32 and fig. 33 and fig. 34 for exploratory investigations. No formal development iteration cycle was adhered to but from the workbook, oscillation between goal oriented and exploratory work was present. So creative and software development iterations were of flexible length and no clear recording of start and end of iterations were kept. This was not necessary because the team consisted of one person. The switch from adding requirements to implementing a section of code to verifying the code to exploring the work was fluid and interdependent.

No explicit team communication was needed for this project because the artist was the developer and the tester. Progress was tracked during the project on a public blog to provide feedback to my supervisors, who are also defined as stakeholders in this project (Grotepass, "maiatoday"). In addition skype screen share sessions provided more direct and visible feedback session for this project.

All the code was tracked using git as source code control system and the project was uploaded to GitHub repository providers (Grotepass, "maiatoday (maiatoday)"). The initial decision to use source code control was taken for backup reasons. This decision proved to be useful for providing creative freedom as it allowed multiple ideas to be explored from the same code base without destroying something that was created, by simply branching and making changes on the

branch. All four works are branched from the same starting project and branches to experiment with different colour choices or particle movement choices were easy to create. In addition it provided reliable backup of the work and allowed working on multiple machines seamlessly without losing anything between machines. Using a source code control system supported both the creative process and the software development process. It also provides a platform to allow code re-use among multiple projects and makes the code available for re-mix by other coders. Manovich's principle of variability (Language of New Media 55) is apparent in the multiple branches which were created for each work. All the branches are descendant from a common starting point.

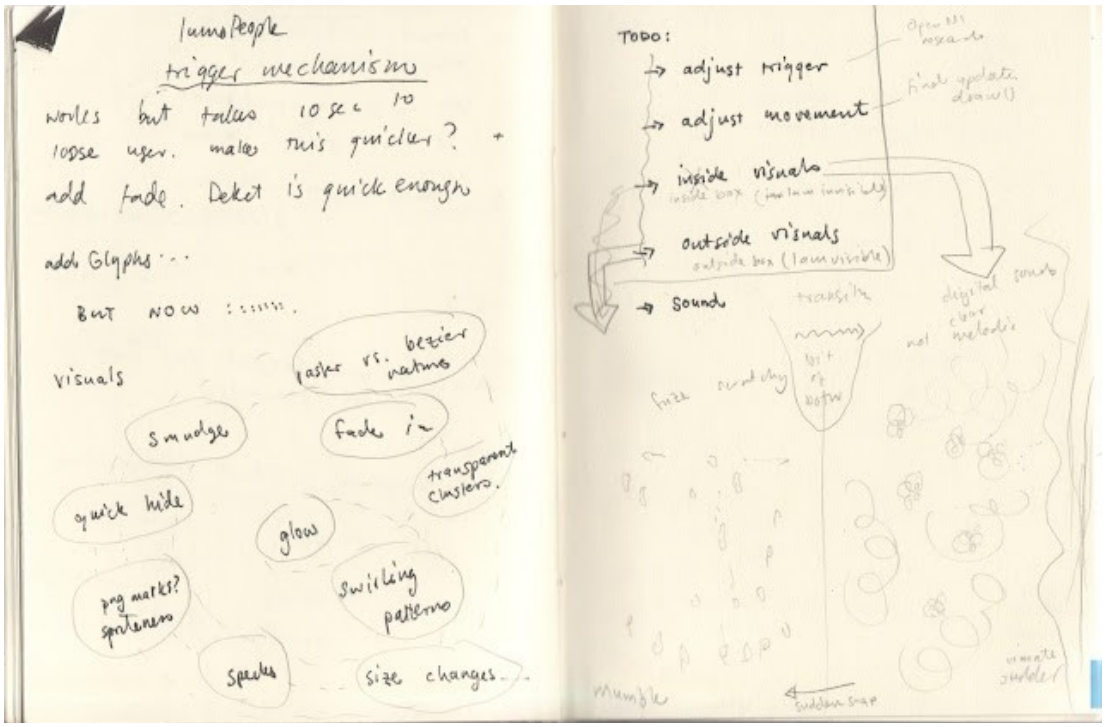


Fig. 32: Goal and exploratory approach in workbook
Source: Workbook scan M Grotepass 2012

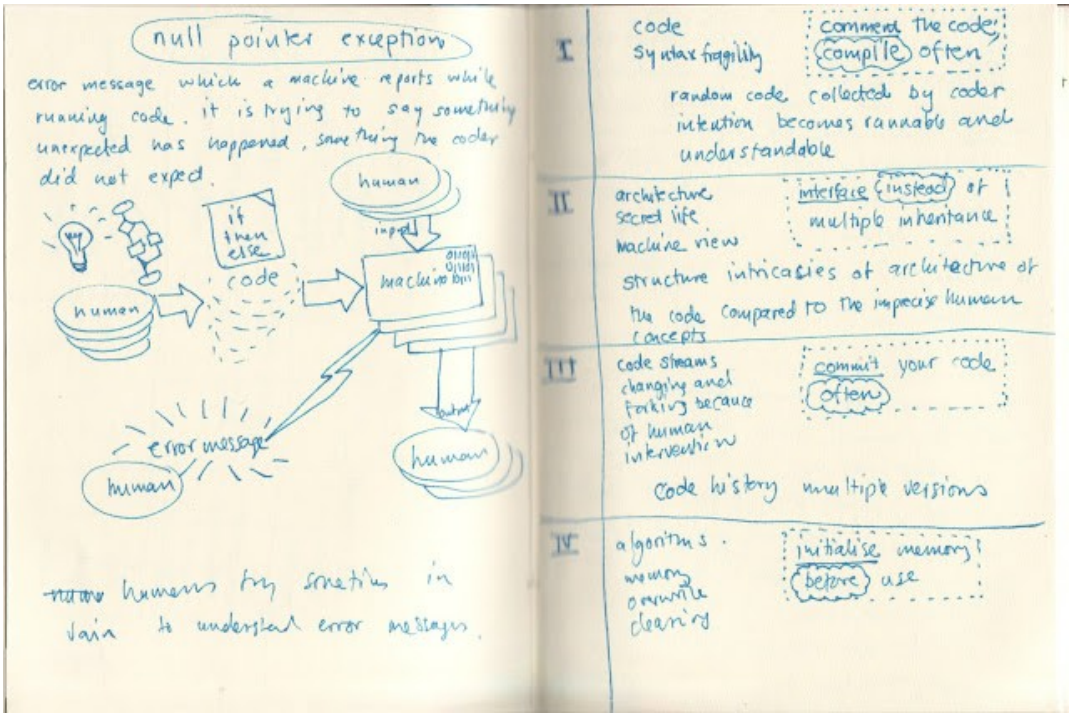


Fig. 33: Goal and exploratory approach in workbook

Source: Workbook scan M Grotepass 2012

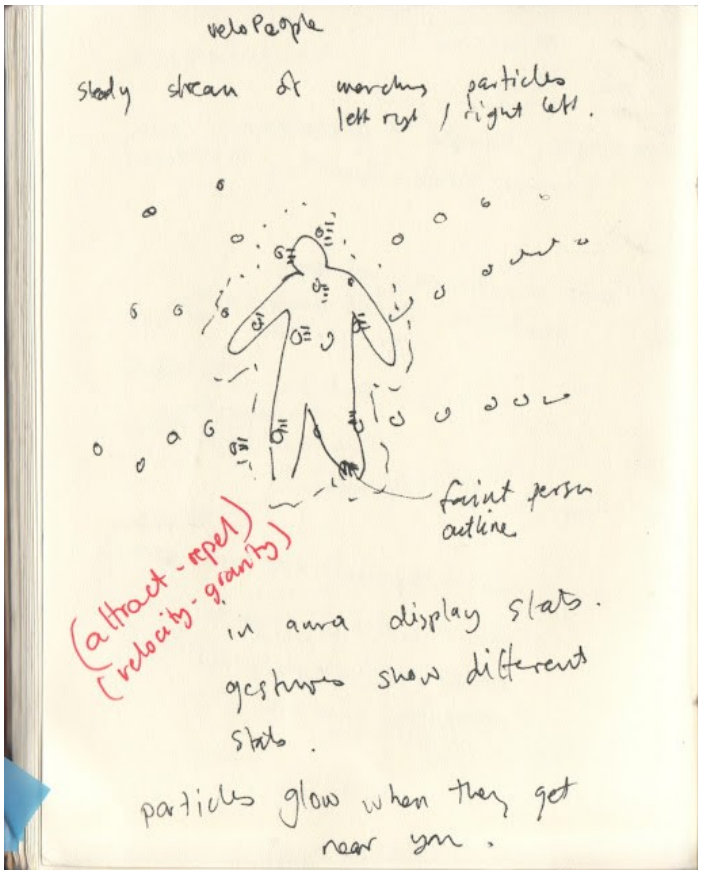


Fig. 34: Investigation of possible visual handling

Source: workbook scan M Grotepass 2012

6.3.3 Architecture

The four pieces were built on similar underlying software structures. A virtual 3D space is created with particles that move and draw within the space. The physical environment is sensed using the Xbox Kinect sensor and this data is fed into the particle system. Each particle moves and draws differently depending on the sensor information.

The particles act as autonomous "paint brushes" with a predefined logic which dictates what will be drawn based on the coded behaviour and the sensor data. The particles are capable of rendering text, drawing a shape or drawing an image. Each particle is also allocated one or more colours. The choice of a particle system was informed by the physical investigation of mark making, which was done as a pre-cursor to starting on the implementation as part of the design stage. It is a way to create multiple active mark making engines which respond to input from the sensors. Using particles is also a way to automate the drawing process so that a complex active work can be created by using multiple modular instances to create the visual layering. Encapsulating algorithms and rules in particles and then allowing the software system to run, applying the rules encoded in the particles and the system, creates a simulation of the concepts in software space. This system creates emergent visuals because the outcome is not known and emerges when the system runs.

On a conceptual level the work explores the human and a machine interface. So a software construct that represents three-dimensional space was created. The human presence in the physical space is detected and projected so that the soft-

ware space and physical space overlaps. Within this overlapping space multiple software particle objects are created. Each particle behaves autonomously according to the laws of motion in the space and according to an algorithm encoded in each particle. The algorithm controls the type of movement and the way the particles make themselves visible. This software constructed space becomes the interface through which the human presence interacts with the autonomous, algorithm containing, particle instances. In the physical space we are surrounded by processors executing algorithms. As an analogy in the software construct the participant projection is surrounded by particles executing algorithms. So this allows the participant to interact directly with the particles and the algorithms. The particles make their behaviour visible by rendering differently. The renderings of the particles are projected from the three-dimensional software space onto a two-dimensional software canvas. This two-dimensional canvas is then rendered and displayed and physically projected on a physical screen. So the rendering process provides the human readable view of the software particles. A digitised version of the participant provides the machine readable view of the humans in the physical space.

A short extract of the implementation of one of the basic classes of this implementation is presented in Appendix II: Extract of DataMote particle class. This object or a variation of this object was used in all four works. From this code snippet one can see the linguistic nature of code. The object name is DataMote, which refers to an element of computer data but also to a dust mote. A dust mote becomes visible when the sun shines on it and it is a particle floating in three-dimensional space. In a similar way the DataMote floats in software space and

becomes visible when a player moves into the space. It is an example of a small module of code which contains algorithmic behaviour and can be re-used. It also shows computer and human language interspersed.

6.3.4 Sound

The sound system in *interfaceInstead* ran autonomously using algorithms which would read samples from disk and play them at random intervals. The sound samples were pre-recorded but the sequence of the sound samples was controlled by algorithms of the soundscape software. The soundscape software used Python and open source libraries provided by Andrew Plotkin (Plotkin). The soundscape software listens on a network port and the visual part of the artwork sends a simple message to the soundscape software indicating when someone is there or when the sub-mode of the visuals changed. The soundscape software adjusts the quality and types of sounds on command from the visual and sensor part of the work. The soundscape implementation is modular in that it can be triggered from any source over the network. It can also be re-used by simply providing different sound recordings.

6.3.5 Testing

Although testing wasn't planned and documented explicitly, different kinds of tests were performed. On the first level, I observed and interacted with each work as I was building them to verify that the software was producing what I intended. This kind of testing approach is similar to the exploratory approach described in chapter 3 although the execution was less formal. The aim of these first level

tests was to explore the behaviour of the system as well as to verify if the state of the implementation was what I intended. This kind of testing was continuous. Small sections were coded, compiled and then tried out, adjustments were made and the cycle was repeated. This kind of exploratory functional testing formed part of the creative process. It confirms the iterative nature of my creative process but is also similar to Agile process recommendations which propose that all process phases happen at the same time during and iteration (Beck 71).

Early stage screen test were done to decide on the translucent screen material. Fig. 35 shows the record of these tests in my workbook. Again the approach to these test was exploratory. It was also accompanied by research on the Internet about other screen solutions such as paint or professional screen materials and commercial screens.

The run reliability was tested by letting the software run overnight and recording memory footprint as the software was running. See fig. 36, fig. 37 and fig. 38 for results of these tests. The memory size was recorded and plotted every minute for about an 8 hour period. The memory size increased when a participant moved in front of the sensor but if the system was left for a longer period the memory foot print did not increase. This test was done to ensure that no memory leaks were present and also to confirm that the works could run for an extended period of time without intervention.

An early test of two works in the gallery space confirmed that the works functioned well in a larger space. Testing with more than one user was performed to see what the effect of one player and many players would have on the system. Tests with multiple Kinects were done to ensure that the infra-red signals from

one device did not interfere with the others and to find out what installation limitations there are for installations using multiple Kinects. This testing was performed in advance before the works were installed in the gallery space.

In general the approach to testing was exploratory and some record of the tests were documented. The testing was less formal than an engineering development project, which has contractual obligations, require. At the time when the tests were performed they weren't seen explicitly as test but rather as creative explorations, or play. In addition the tests were used as a mechanism to see how the work had to be adjusted. From a creative process point of view they functioned as part of both the divergent and convergent cognitive activities because they were used to create new ideas but also to make decisions on which ideas to discard. Even though some definitions of testing views the activity as having a validation function only, the exploratory testing approach allows learning about the system and rapid feedback as a valid goal of testing (Bach, "Exploratory Testing Explained" 7). On reflection on the software development aspects of this project and using Bach's definition of testing, these activities can be recognised as testing phases. This kind of less formal testing is suitable for a project where the artist is the customer, the developer and the tester. More formal testing may be required for projects with diverse team members.

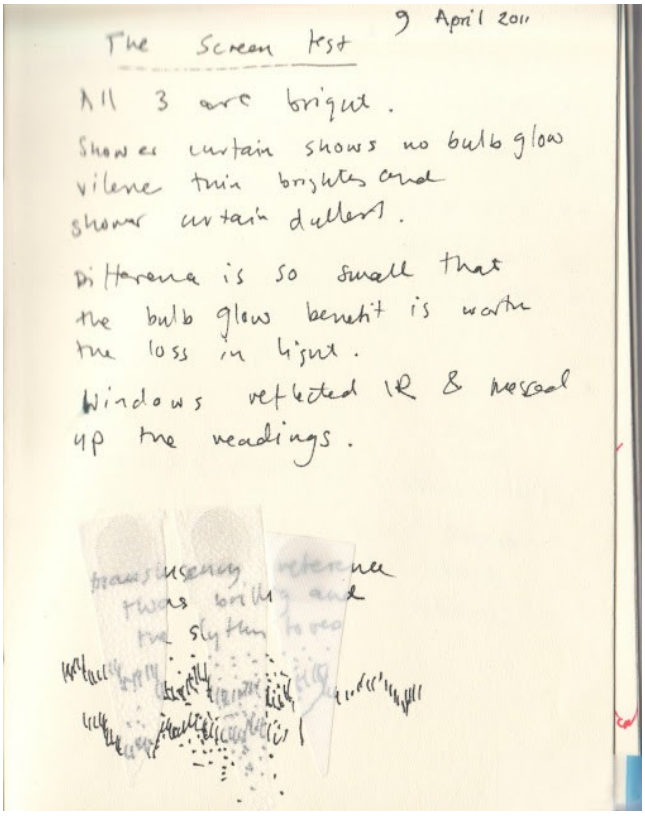


Fig. 35: Screen material test

Source: workbook scan M Grotepass 2012

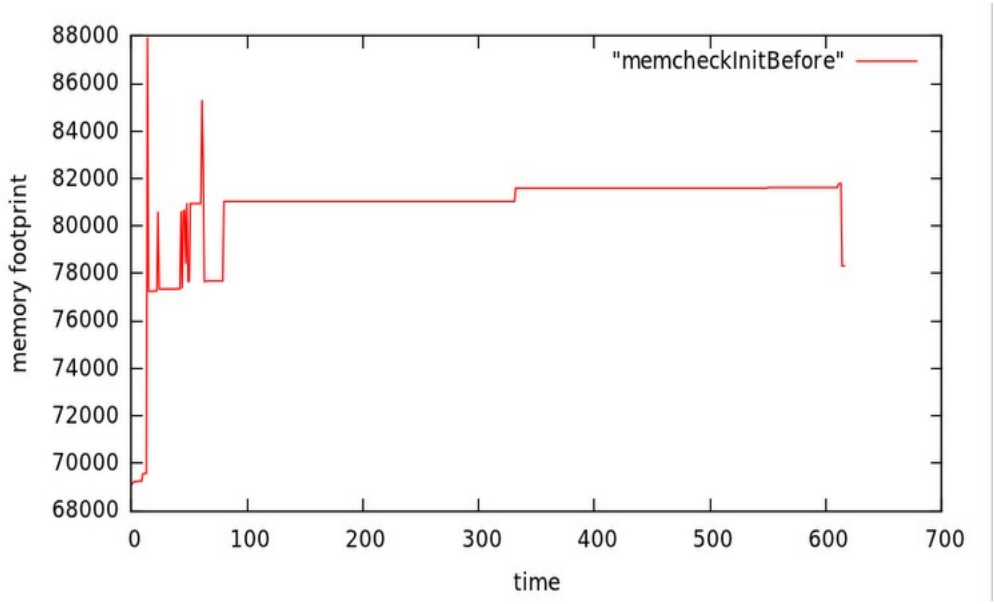


Fig. 36: initBefore memory footprint graph

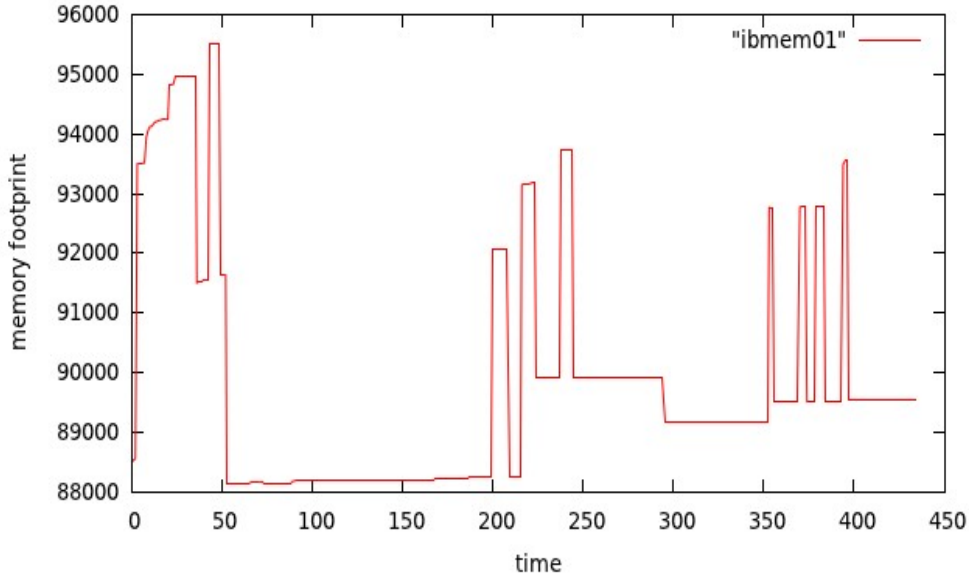


Fig. 37: commentCompile memory footprint graph

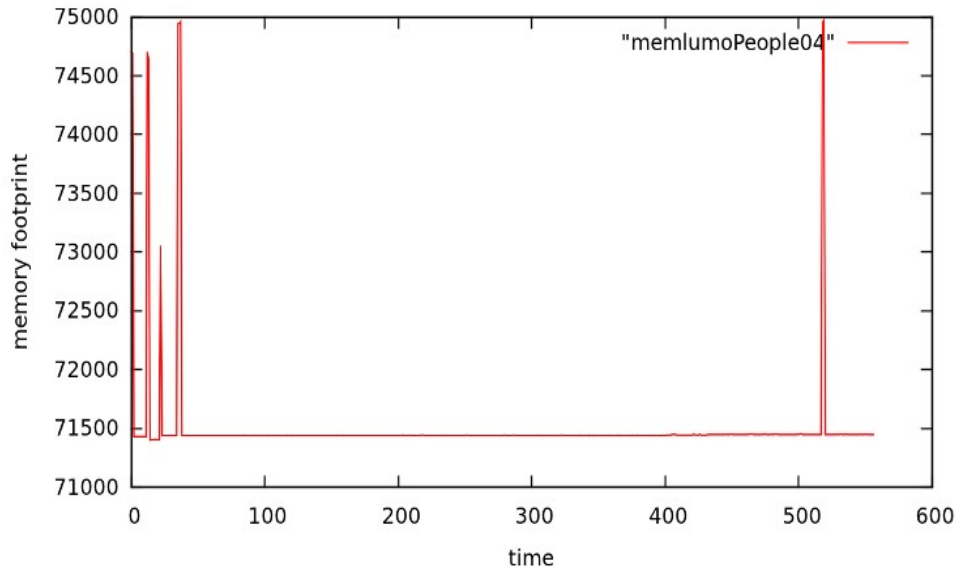


Fig. 38: commitOften memory footprint graph

6.3.6 Reliability requirements

The intention of this installation was to create a technical solution that would require minimum interaction by gallery staff. At best the gallery staff would just be required to switch the projectors on in the morning and off in the evening. In addition if some form of power failure occurred the machines would have to boot by themselves with no intervention. The artwork would run in fullscreen automat-

ically. In order to do this a live-boot USB flash disk was created. The flash disk would run automatically when the pc booted. In addition the system would run whether the projector was on or off so that it could continue to run even if the projector was off. Furthermore the system had to be reliable enough to run without crashing for the duration of the exhibition which was 3 weeks. This installation succeeded in these requirements. One system had to be restarted, but this was due to a fault in the multi plug which powered the system.

6.3.7 Installation/deployment

As this installation required multiple computers and the budget for this exhibition was limited, the specifications of the machines were not known ahead of time. In addition the machines available could not be reconfigured for the exhibition. In order to do this the artworks were installed on external USB disks so that the hard drive and configuration in the machine was not affected. This was done using a live-boot USB disk which would detect the available hardware at boot time. A further benefit of the live-boot USB solution is that it can be shared with people. To view the works, the disk and a Kinect sensor is plugged into any computer and the computer is booted from the disk.

The screen resolution capabilities of the graphics system of each machine was not know either. The projectors supported at least 800x600 but once the particular machine and projector combination was started up this resolution was sometimes larger. Also different machines were capable of running at different speeds. To accommodate resolution and processor power, the particle count for each artwork had to be adjusted. As it was difficult to re-compile and re-install the art-

works on the USB disks in the gallery, some parameters such as particle count was read in from an external file so that this could be adjusted for each work in the gallery when the work was installed.

The installation in the gallery was done in such a way that the machines were not accessible and visible. Each machine was installed behind a partition. This protected the machines from tampering. This also meant that the machines were less accessible if a problem occurred. This installation choice also required a live-boot solution which did not require outside intervention.

7 Conclusion

I have looked at the software artwork creation process with multiple lenses: the creative process aspects, the software development process angle, the software as art medium perspective, from the point of view of the artists from interviews and by reflecting on my own practice. Using software as medium requires interaction with computers and technology. The medium has its own peculiar characteristics. It is transient and mutable and gives the artist control of the behaviour of the machine through the code, allowing the artist to create active, changing artworks.

Creating the code for a software artwork is a software development process. The engineering field provides insights on software development process for non-art projects. Since the creative approach of artists can vary, a software development process, which can accommodate changing ambiguous requirements and is adaptive to different cognitive styles, was investigated. The Agile family of development processes has these characteristics.

From the Agile development process I learn that using an iterative development cycle with a feedback mechanism which allows the process to adjust to changing requirements is a possible solution. The feedback mechanism takes the shape of demonstration of working code, discussions with team members and exploratory testing of the project to allow the artist and developer to learn how to adjust the work.

Feedback and testing can improve communication and support collaboration. Collaboration is important for the success of a software project but is also a key recommendation for a successful creative project.

Since creating a software art project also relies on a creative process, I investigated the research of the creative process. Writing code is a cognitive activity, so I narrowed my investigation of the creative process to the cognitive psychological field. From my readings I learn that the creative process can also follow an iterative cycle. The cognitive activities oscillate between a divergent ideation phase, where new possibilities are generated, and a convergent evaluative phase, where the possibilities are evaluated and the process adjusted. Creative processes that are adaptive and emergent allow novel ideas to take shape. Creative projects can be approached with different cognitive styles, ranging from a goal oriented approach to an exploratory approach. If a software development process is to help the artwork creation process, it must support the creative process. Recommendation for tools that support the creative process provide guidelines. A software development process must accommodate different cognitive style approaches. It must be flexible, adaptive and simple to use. It must alleviate the barriers to collaboration and it must not hinder experimentation.

The Agile process values and principles are aligned to these recommendations: it provides tools to improve collaboration, it is adaptive and it can accommodate goal oriented and exploratory approaches. It values being responsive to change and proposes self organising teams. The Agile process has to be tuned to match the artists' intention and some recommendations are not appropriate for digital art projects. Automated and scripted testing may not be justifiable in the context

of digital artwork. Also a rigid approach to following iteration cycles may be counterproductive because it may interfere with the creative iterations.

There are general engineering practices which can also assist the software artwork creation process. The characteristics of software as medium highlight its mutability and variability which results in multiple versions. Engineering practices provide tools to manage software streams. Source code control is such a tool. Source code control allows the artist to keep different version and allows the artist to access any branch of exploration without destroying existing work. Source code control allows code to be accessible to the artist and the community. This supports re-use and collaboration. It is an automatic documentation of the software code development process and can function as art making documentation.

Another engineering practice which can assist the artist is testing. For testing to be useful, though, it has to integrate with the creative process and not be seen as a formal activity which is used to find deficiencies in the software. The exploratory testing approach, where the focus is on learning about the software system, integrates with the ideation phase of the creative process. The exploratory testing approach can also be used in an evaluative way to refine a project by assisting the artist in seeing how the software needs to be modified.

In answering the research questions, I see that there are engineering practices and Agile practices which can help artists when they create software artworks. Reflecting on my own practice, I can confirm that iterative cycles, source code control and exploratory testing were natural and useful activities for my process. From interviews with artists I learned that art as medium and the conceptual as-

pects of the making process were the focus of the interviewees. Using software as medium changed the process and thinking but the artists are thinking and discussing the medium and the conceptual aspects, not the detail of the development process. The complexity of implementation can become a distraction. Whatever support engineering processes provide to the artist, must be balanced by the individual creative process of the artist. The development process should assist the artist to focus on the ideas that sparked the project and allow the artist to keep the concepts that interest him in mind, by providing momentum and support.

From the interviews, process activities which occur in the Agile process were observed. Regular analysis/reflection which informs implementation and testing which informs the subsequent analysis and reflection can aid artists to maintain a balance between conceptualisation and implementation activities as well as the cognitive and creative activities associated with each process phase. Reflection and testing provides a way to explore the limitations of the tools and the medium and allow the implementation to push the boundaries of the tools and the medium. In this way the Agile process can give a clearer understanding of the software development activities and offer insights into an artists process which will allow the artist to adjust both the Agile process and the creative process to the goals of the project. From the interviews the process should stay adaptable and be able to follow technology changes or adapt to different process needs of different projects.

Software and computers are tools but are also being used as a medium by artists. Hayles says: ”the computer is not so much a machine as it is a mind ampli-

fication tool and different kind of expressive medium" (My Mother Was a Computer 60). Software development processes can be seen as tools that support the development process and can affect the way artists think and work but should not undermine what Lovejoy sees as "the most fundamental aspect of art-making" which is "the coherence of the artist's conceptualization process" (31) .

8 Appendices

Appendix I: Contents of disk

File or Folder	*Contents*
Invites/	Folder with the png images for the exhibition invites
sourceCode/commentCompile	Folder with source code for *commentCompile*
sourceCode/commitOften	Folder with source code for *commitOften*
sourceCode/initBefore	Folder with source code for *initBefore*
sourceCode/interfaceInstead	Folder with source code for *interfaceInstead*
website/	Folder with the website accompanying the exhibition
nullPointerExceptionCatalog.pdf	PDF file of the exhibition catalogue
nullPointerException-commentCode.mp4	Video documentation for the work *commentCompile*
nullPointerException-commitOften.mp4	Video documentation for the work *commitOften*
nullPointerException-initBefore.mp4	Video documentation for the work *initBefore*
nullPointerException-interaceInstead.mp4	Video documentation for the work *interfaceInstead*
MGrotepass2012.pdf	PDF file of this thesis
MGrotepass2012.mobi	E-Reader version of this thesis
MGrotepass2012.epub	E-Reader version of this thesis

Appendix II: Extract of DataMote particle class

```
void  DataMote::draw ()
{
    if (label == 0) {
        drawOutside (); // this particle is not inside a participant
    } else {
        if (label == frontUser) {
            drawInside (); // this particle is inside the front most user
        } else {
            drawInsideNoFront (); // this particle is inside a user
        }
    }
}

void DataMote::drawInside ()
{
    bool drawSquare = false;
    float f = 2;
    // I am drifting aimlessly or not if flipped
    float dist = getConstraintDelta ()/maxDistWidthSquare;
    if (dist > 0 && dist < 1) {
        myAlpha = insideColor.a;
    } else {
        myAlpha = ofLerp (START_ALPHA, STOP_ALPHA, dist);
        drawSquare = true;
    }
    ofSetColor (insideColor.r,insideColor.g,insideColor.b, myAlpha);
    ofFill ();
    if (dist>=0 && dist<0.01) {
        addVelocity (ofPoint (ofRandom (-f, f), ofRandom (-f, f), ofRandom
(-f, f)));
    } else {
        setVelocity (ofPoint (ofRandom (-f, f), ofRandom (-f, f), ofRandom
(-f, f)));
  }
    if (drawSquare) {

        ofRect (getX (),getY (),_radius,_radius);
    } else {
        ofCircle (getX (),getY (),_radius);
    }
    if ( (fadeCount <= MAX_FADE_COUNT) && (fadeCount > 0)) {
        fadeCount--;
    }
    if (fadeCount > 0) {
        drawOutside (outsideColor,
outsideColor.a*fadeCount/MAX_FADE_COUNT);
    }
}
```

Appendix III: Email interview with Brogan Bunt

Brogan Bunt brogan@uow.edu.au
11/23/11

to me
Hi Maia,
Sorry, my website is in a deplorable, neglected state at present. Just thought I'd send through the semi-technical info on the Loom exhibition, a few of the images (low-res), and a wordy paper I gave at ISEA this year on the project. Be interested to see your cross-hatched work.
Cheers,
Brogan

Associate Professor Brogan Bunt
Head of Postgraduate Studies
Faculty of Creative Arts
University of Wollongong

From: maia grotepass [mailto:maiatoday@gmail.com]
Sent: Wednesday, 23 November 2011 7:19 AM
To: Brogan Bunt
Subject: Re: code and creative process

Thank you for the responses, Brogan. From the feedback I got from the people looking at my recent works, I agree that people don't really care about the software and the process, except for artists here and there who are engrossed/entangled in the code themselves and other programmers who don't always understand why I am not coding something "usefull" even though they are usually too polite to express it like that :)

Are you referring to your works titled Loom? I had a look at some of them on your website. I find the shift between the algorithmic/geomteric aspects to organic almost random and natural looking elements mesmerising. A similar thing happened in one of my pieces where I was looking at cross hatching as a traditional drawing technique and then using algorithms to subtly change and repeatedly overlay the hatches. I will read through what you have written about theses works too, as I think they relate to what I am busy with at the moment.

Maia

On Tue, Nov 22, 2011 at 8:37 AM, Brogan Bunt <brogan@uow.edu.au> wrote:
Hi Maia,
Thanks for getting in contact. A quick response to your questions:
1. Coding requires a kind of obsessive discipline, but I don't find this too removed from my other modes of creative practice, which also tend to involve rules, systems and the like. I don't really adhere to a directly expressive paradigm. The personal is kind of hidden within the ruse of formalism.

2. In my experience nobody really cares about the software or the process of programming. It may be important to me, but it is not something that I can show to an audience except in some kind of allegorised, indirect manner. So one of my recent exhibitions used recursive polygonal subdivision as a means to reflect upon dimensions of computational labour. The work appeared simply as printed algorithmic drawings – with no effort to exhibit the underlying code engine. Something of the nature of code logic was made visible, but via the visual patterns rather than the code itself. I'm also interested in placing code in odd contexts, in designing programs that intersect with non-computational domains. This involves exploring links to traditions of instruction-based conceptual art.

Your website looks great. Good luck with the Masters.
Cheers,
Brogan

Associate Professor Brogan Bunt
Head of Postgraduate Studies
Faculty of Creative Arts
University of Wollongong

From: maia grotepass [mailto:maiatoday@gmail.com]
Sent: Sunday, 20 November 2011 10:08 AM
To: Brogan Bunt
Subject: code and creative process

Hello Brogan

I wanted to let you know that I found the writing for your P.h.D (Risking Code - Software art - dilemmas and possibilities) very informative. I am an artist, who uses code as medium, and I am busy completing my M.A at the Digital Arts division of the University of Witwatersrand in South Africa. In particular I used the dilemma of visibility as discussed in Chapter 4 of you dissertation as a starting point for my practical installation art.

At the moment I am in the process of preparing the written research part of my studies. I would like to get some opinions from artists about how they think using software has affected their creative practise.

The questions are:
Has using software as medium changed your creative process? If so how?
What problems or benefits do you experience when creating an artwork using software?

I would appreciate any views you have. Also if you know of anyone who would be willing to discuss these questions with me, I would really appreciate it if you would pass on this email or send me their details and I will contact them directly.

Hoping to hear from you.
Maia Grotepass
www.maiatoday.co.za

Appendix IV: Transcription of Skype Interview with Joshua Goldberg

Transcript of skype conversation with Joshua Goldberg (http://goldbergs.com/)
23 November 2011

jg: Maia
mg: hi
jg: hello
mg:How are you
jg: I am well, it's always nice to hear a South African accent.
mg: ok, I am on the netbook so you might actually get a video too.
jg: ah well possibly we'll see, as long as it doesn't interfere with your bandwidth cap.
mg: no that's fine
jg: do south africans network subscribers mostly get a bandwith cap still?
mg: yeah but I've managed to get an uncapped now, so that's ok.
jg: oh congratulations. I think bandwidth caps are the devil, to me there is just no reason for it. It punishes everybody because of the 3% of the people that are torrenting and any business should assume a little tiny bit of fraud, there's always going to be fraud.
mg: and there are ways to sort that out that torrenting stuff
jg: I know it's so stupid
mg: ok
jg: allright so shoot, ask the questions.
mg: well, I don't know if you spoke to Tegan at all but I sort of come from an engineering background and then I got really into art stuff so little bits of me are still with the engineering process and little bits of me are with the artistic process and so what I am interested in as an artist how do you find that your process changes if you are using software that could possibly well that has technology and it actually works differently or has different requirements sometimes of you.
jg: ok I have a really good answer for this question, probably the single biggest difference between an artist who uses conventional analog non-electronic methods of creation, for instance a typewriter, or a paintbrush, and an artist who relies upon technological change to bring an increasing ability to manipulate the ideas and create the ideas that they want is this: it's the responsibility of the digital artist to hate his tools, to hate them, to loathe them and to constantly be thinking about the ways that these tools are restricting one's output. If you know .. if you don't understand that your tool is deficient you will not be hitting one of the major things that an artist should hit which is to push boundaries. To look at what's possible, to look at what's next. what is the reaction, what is the unique reaction to this world that an artists perspective brings on. If you do not, as a digital artists, stay constantly aware of the limitations and drawbacks of the methods that you are using, you're derelict. you're not going to be doing.... You are not hitting the touch stones you should be hitting as an artist.
mg: ok but that's kind of true for other tools as well, I mean analog tools are also, have their limitations. I think it is the responsibility of an artist whether whatever the tools are to kindof go to those boundaries.
jg: ok but I think that it's easy to say well yes of course artistic tools, analog tools have limitations, but give me an example that you are thinking about and I'll show you how I disagree.
mg: ok well if you take something like paint,
jg: ok so what innovations what . lets say that you were painting five years ago, ok. even more, let's say you were painting fifty years ago, how has painting, how has the technology behind using a paintbrush made it so that you are no longer, your work is no longer viable.
mg: that it's no longer viable?
jg: viable. interesting. there is plenty of work being done by artist who stopped innovation with respect to their tools, that is just as important and just as vital, now as when they were making work in their twenties. That you can point to a whole host of painters who are still working in this way, whose work is still important, even artists who started to begin to think about technology, as David Hockney, Hockney's work is still important. The thing about artists who use digital tools is that it's a tool ??,not only the point that I'm making about, now you know, you used MSPaint 20 years ago, using MSPaint now, unless you have a very very good ironic artistic justification for doing it, you are out of your skull. and you've no idea. your work is utterly irrelevant. On the other hand. and the

second part of that is that there's. I got a little confused, this is the problem with doing this aurally. The thematic concerns of a digital artist may also evolve but what is much more important is that the digital artists know the limitations of tools that they are using to create to create this, Ah file formats that's what I was talking about.

The problem with the digital artist who is using MSPaint now as they did 20 years ago is not just one of limitations of the tools that they are using but it is also file formats. We will have a point sometime in the next ten years where .bmp's which is the native file format of MSPaint is not going to be technically readable by a machine that is coming straight out of the factory. I know people who were making CDROMs and CDROMs as an art-form in the mid 90s and there isn't a single computer that's been made in the last 8 years that can read a CDROM that was authored by voyager for OS9. It's almost impossible as well to take the CD that you wrote for windows 95 and have it run under WIndows 7 machine right now and these people were very very dedicated to their artwork.
mg: so you're saying just because, you're saying that they didn't realise the limitations of their tools and so they made something that we can't access now.
jg: This is a two part, I'm sortof talking about two things at once. First off file format no they didn't and now I think we all need to realise this on a mass scale is that art that we are making currently with computers is ephemeral and temporary. There's no law that says that jpgs need to be an understandable file format in 50 years. And it won't be you know, CDROMs disintegrate hard drives die. You're making art for the decade, you are not making art for the ages.
mg: there are some traditional artist who make art out of lumps of butter and those things didn't last either so
jg: yes and all the joseph conrad mixed media stuff is fading and looking a little bit worse for wear these days, totally understand, but it's much more the case, it's much more a pressing issue with art which is made in a digital domain.
mg:, yeah I agree with you there. But that's the nature of the medium.
jg: That's the nature of the medium. But this is a little bit different from the hate your tools thing. The hate your tools thing is, are you making something that the creators expected people would make with the tool, or are you transcending the limitations and the intention of that tool to push the envelope further.
mg: ok
jg: so when you have someone who makes a Photoshop, or how makes a MAX/MSP or who makes a Flash, who says about an artist, wow I can't believe that they are actually using this tool that way, I guarantee you the artists they are talking about, looked at what people were making with the tool and what they could do with the tool and said, wow this is so stupid, why can't I do this, and thought about a way to use it differently. And I actually think that, that is a responsibility, it is a big responsibility, otherwise toolmakers become complacent. If you don't make your own tools, and you don't use tools that others made in ways which are surprising and/or repellent to them, then you're not really making art which is standing on it's own. you are reinforcing a mean.
mg: OK, so you're just perpetuation what the toolmakers wanted you to do with their tools.
jg: That's correct and that's not really art. That's not really art. You're following the footsteps, you're making cookie cutter work.
mg: ok. I see what you are saying. So even if someone writes a tool which is really flexible, because they want people to make art with it, then you've still got to push it to it's limits.
jg: right, there's no such thing as the perfect tool. There is no such thing as the perfect tool.
mg: Yeah well traditional tools aren't perfect either, traditional tools have their limitations too. You don't get a perfect traditional tool either.
11:03.2
jg: but also If someone says, I love the oboe, I will dedicate the rest of my life to the oboe, that's not disturbing, people consider that respectful, people consider that disciplined, and worthy of appreciation. but if someone says Wow, I really love Photoshop 4.0 I'm gonna spend the rest of my life working with Photoshop 4.0, there's something up with that.
mg: yeah it does feel a little bit sortof like copping out in a way.
jg: that's right.
mg: ok I think I agree with you there.
mg: so you're actually saying that if you are using digital tools that there is more of a demand on you to push the tools ...
jg: to break the tools, to transcend them, the shorthand is to hate them, this is not a new sentiment, this is the best advice that I got when I went to grad school from a very smart man named jaron

lanier (http://www.jaronlanier.com/), who was the inventor of the old-shool goggle and glove interface.
mg: I can google her,
jg: him, I am giving you his name now (good god L A my poor keyboard NIER)
mg: ok I got it, I"ll check it out
jg: good, next question
mg: I think you sort of answered part
jg: did I cover all of it
mg: I am sure I could carry on for a longer time, I picked only two questions cause the previous round of questions I had with people were too long. The next question really what are the positive things. I mean you are saying one of the things about the digital tools are that, or the question is just are things that you.

jg: oh I'll tell you the positive things, the positive things are easy. So if I was a painter working in the 1950s and I decided that what I really wanted to do was make a painting which looked different depending on what time of day it was, I'm dead in the water. I have the wherewithal to think about the limitation of my tools but I am not living in an age where I can say Oh ok, so how do make something like this alright well I need to find a sensor that tells what time of day it is or what the humidity is about this. and then I need to figure out some sort of shader that I can apply to do colour transformations efficiently based on the idea, although again, this is kind of the same answer because the path from outlandish idea to possible realisation of this idea has shrunk drastically in the digital art making age but on the other hand those paths become cliched much quicker.
mg: ok I see what you are saying, Just because the tools are super powerful it actually makes have to think up even more outlandish things to be pushing the envelope all the time.
jg: Right and this is not to say that artists before didn't have outlandish ideas. They just didn't they just weren't in a situation where they thought it was possible to act on these outlandish ideas. and the ones who did try to act on the outlandish ideas it would consume their life and they were regarded as crackpots.
mg: ok I'm going to switch to some other questions that I have which might be interesting. I have been looking at some of the stuff you were working on and you are using MAX/MSP. I haven't used it much but the way I understand it, Actually what I am doing is I'm relating it to an installation process that I went through recently and there's like a phase where you are trying things, to make sure things are working which is kind of like and exploring but also like a testing phase and at some point I had to decide which of the variables are going to go fixed and which of them are going to be flexible. because to change location I had to decide which of the variables that I am working with are going to be affected by the location and kindof had to make a split so at that point, for me it was a more step by step process cause I couldn't compile I couldn't change the code once I was in the space that I was going to be in so I had to do the split ahead of time and find a way to change that. I think that with MAX/MSP you can change things on the fly much easier. So it feels like there are bits that are testing and fixed and pieces that are flexible. And to figure out that's the one thing. What am I asking?
jg: Ok Well can I tell you something which is sortof related to this. Can I tell you a little anecdote? 17:03.0
So let me tell you about a piece of mine. My website is just so awful there's a piece of mine that is called ???"the south star", and it's a very large LED sculpture. And it uses sound levels and it plays with the sound levels but the interesting thing about it is that we used sound to gate the power of LEDs with a custom VU meter circuit, you know what a VU meter is, the thing that makes the needles jump up and down on the stereo when you're playing music, so this is a custom VU meter it gates up to 10 LED panels of power. The power that we are gating to these panels we're stepping down alternating current with triads. We're not using traditional transformers to gate direct current, we're gating alternating current and we take advantage of this by having half of the LEDs of the panel, the blue LEDs on the panel, be on the positive phase and half of the LEDs on the panel be on the negative phase. When we first started building this were like it's really beautiful because it's really flickery and you know it can't be video taped, but then I put on wandering star by portishead and I ran it through the circuit and we were blown away because that big dominant base note, the wandering star's, is very close to 120 Hz and the thing about 120Hz is 120 Hz is a multiple of the frequency of the alternating current ground in the States.
mg: ok so you get like a harmonic thing going there

jg: yeah you saw a harmonic, so it turned out that when we were very close to a harmonic of the line frequency
mg: of the power source
jg: we would get all one colour fading into another colour fading back in. It would go all blue and then it would go all orange because it was sampling only, it was sampling the alternating current at that time. And so I have a whole piece built on playing with these harmonics and you begin to see the problem now right. I can't show this outside of the states.
mg: yeah unless you cook up some electronics to fake it out.
jg: because , exactly exactly. And then everytime we actually have start cooking that it always looks a little different, it doesn't really look the same and because of this we have a piece that really can't be shown internationally.
mg: that's true but the question is are you going to focus on trying to get it to work everywhere or are you going make something where part of it is that you can't show it everywhere.
jg: well both is the answer, actually. they're both interesting. it actually opens up. That issue opened other ideas that I was using to think about artwork, and I mean. There's other things that are like this. I am currently, in performance I have an algorithm that I use for digital feedback, which is something that I am really, I completely love, I think it's a transcendently beautiful thing and it only works in a very specific way of manipulating matrices on the CPU and not on the GPU so I am to a certain extent limited in the resolution that I can work on this thing because it looks a certain way when I'm running at 640x480 and I tried to actually on to shaders and all of my methodology and all of my programming changes and it looks different. And I but basically it means I have to take a chunk of my artistic practise and seal it in amber if I want to be able to reproduce work of this nature.
mg: or you can just live with the fact that you can't reproduce it
mg: maybe it will help if I give you an idea of the kind of research that I am looking at. What I'm doing is I'm comparing some engineering processes and comparing some creative processes and of course you can't apply all engineering processes to artistic process because that would just.
jg: unless it's a coherent artistic strategy. I mean I know folks who do, who think about their work as engineers as and what they make as a by product of that is tremendously beautiful art. And it's just a matter of framing.
mg: but the thing that I see from my experience with software development is if you do software development from an engineering point of view is you have a very clear or you strive to have a super clear idea of what you want in the end and work towards towards that, whereas people that are following an artistic process they kindof changing what they want because they are trying to see exactly what the limitations of the tools are. So it's like the starting point is different. So you think that some engineering processes you can't map because you need to be able to explore.
jg: and you also need to have faith in your tools as an engineer, your tools are, if you can't believe the data, this is a fundamental difference in art and science. if you can't believe the data that you get with art, that can be an artistic method. But with science you have got to spend of your time making sure that the data that you've acquired is legitimate. You can't make a precious point out of bad data, it's just bad science. and the analog of bad science and bad art are quite different.
mg: but still there are some nuggets of engineering process, something like source code control which if you think about the reason they have it in the first place is to kindof make sure that the processes are repeatable but if you are busy exploring it actually it has the opposite effect, you can use the tool in the opposite way because you can explore so many more possibilities because you can capture snapshots. so it's a ??? engineering process.
jg: it sounds to me that this is a question more of how have the impulses and the responsibilities of an engineer changed in the distributed era and that's a another interesting question and one that I am not nearly as capable of answering. are you asking me to talk about the differences of artists as engineers in this format.
mg: no I'm just actually looing at the artists perspective, because engineers there are lots you can ask but the artists view on their process and how the digital tools have changed that there aren't that many people talking about that.
jg: oh there should be
mg: and I'm trying to look at that and comparing that to the engineering processes, Am I being vague...
jg: .. there should be more artists thinking about, coherently talking about why their tools are deficient, something more in depth than , wow this program sucks. and the. I think that we are also at the beginning of artists. I think we are in a situation now where artists need to. actually Part of

being an artist in pre-technological artistic practise was knowing a certain level of engineering that was needed for you to be able to do the work you wanted to do. So if you're a composer you know how to tune your piano, you know how to make sure that the notes that you are playing are really notes that you are going to get or in like the ramp up in early 20th centuary ramp up to digital composition where the guys are starting to work with alternative tunings, Messian (http://oliviermessiaen.net/) harry partch (http://www.harrypartch.com/) these guys who thinking as engineers about ways that they could change the music that they were writing and making as artists and these guys are kindof precursors to the digital artist whose responsibility it is to hate the tools.
mg: so they kind of just changed their tools.
27:12.2
jg: or Yves Klein I keep on thinking about Yves Klein (http://www.yveskleinarchives.org/), because I'm using painters as .. Yves Klein based his career around a singular accomplishment which was a synthesis of engineering and creativity. It was to make this specific blue. So he knew enough engineering to think about pigment mixing and how to replicate that process reliably.
mg: so maybe if things are changing quicker and much easier you can just fall into a trap of just using the tools then it almost demands more and engineering approach of the artists than before.
jg: I think so, I think to be a really mature digital artist you have to be unafraid to be an engineer. They used to be to a certain extent mutually exclusive professions but I think that it's not the case anymore. That with this easy access to differing work flows and easy access to differing paths everyone is being kindof forced by circumstance to become a little bit more of a dilettante.
mg: ok then it's just a question of which bits of ... engineers are really good at writing up their processes and figuring out which parts are cool and which parts work
jg: .. and artists are not
mg: the thing is you can actually learn from those processes but if you were to apply all of them then you would end up being an engineer and not an artist. (hehe) I mean you have to actually, the idea is really I think you have be able to push, well as you said "hate your tools" so you need to be able to push the boundaries rather than gold plating and getting it perfect.
jg: right Maybe that's the difference between a great artist and a great engineer, which is that a great artist knows when it's, when the striving for perfection in the case of this action is unimportant. You know when to stop. you know the Beckett quote right (http://en.wikiquote.org/wiki/Samuel_Beckett), the try again fail again fail better.
mg: Yeah, I think I have yeah
jg: He elaborated on it a little bit more he said at one point he said that it's the responsibility of the artist to fail. to fail over and over again in the sense you got a perfect idea in your mind, you're shooting for that perfect idea, you're not going to be able to achieve it if you spend all of your time if working on getting that one thing perfect you're never going to create anything so it's your job to continually know how to fail at hitting that point. whereas for an engineer to take on to take that mindset is suicide. Get it right do it right, get better at doing it right.
mg: yeah ok so there's like a fundamental mind shift even if some of the processes and tools could be used.
jg: yeah I think that's the big difference between and artist mindset and the engineering mindset in the digital age. That it's not the artist's responsibility to get it perfect just to think about new viewpoints, new ideas, new comments, new workflows, whereas the engineers responsibility, true responsibility is you have to get it perfect.
mg: ok that's kindof a nice summary, that's given me quite a bit of stuff to work with. So yeah I can't think of any more coherent questions now because this is a whole lot of ideas that have been stirred up. so I think I"ll chew on it a bit.
jg: am i your only interviewee who said lets do it via voice because it's silly to type this stuff out.
mg: yeah I have been trying various ways to get people to talk to me so I am very happy that you are actually talking to me, I had questionnaires before and I had email conversations and I've had form posts all the possible ways that I could figure out ways that you can get artists to talk to you. So this is very nice that I can actually speak to you.
jg: if you transcribe all or part of this mail me a transcription of it.
mg: yeah
jg: thank you very much
mg: I'll let you know. I'll keep you in the loop with what I am writing so you can see that I don't misquote you.
jg: cool
mg: so I think that that's enough for tonight thank you very much, I really appreciate your time.

jg: you're very welcome.
mg: and I'll keep you updated.
mg: bye
jg: bye

Appendix V: Skype interview with Pierre Proske

Skype Interview with Pierre Proske - 7 March 2012 12:20 UTC+2

mg: Hi Pierre can you hear me?
pp: Yeah I can, Uhm is my audio ok
mg: yeah that's fine you're fine
pp: uhm yeah sorry for taking so long to kinda get back to you, I've been really busy and I am not on skype very much and of course this time difference makes things a lot more complicated.
mg: yeah no it's absolutely fine, that's fine, whatever works is great for me.
pp: so I can't actually remember your original questions so if you have them with you, or could you sortof
mg: yeah they really simple it's just how has using software as a medium for developing and artwork or an art piece, how has that influenced your process? has it influenced it at all? and then maybe if you can think of some benefits or some problems that you think are specific to using software as a medium?
pp: yeah, i think uhm the difficulty with that question is, for me at least, is that maybe, maybe I wouldn't have felt as empowered as an artist if I didn't have the sortof computer skills as opposed to vice versa. because I guess, when I was growing up I like, .. contemporary art something that was the furthest away from my mind, like it was really a very very foreign and very distant thing. and i guess I sortof discovered it later on when I got older. and yeah really in a way I kindof stumbled across it by accident when I suddenly realised that my interests were converging on. that actually I think because I started reading some material and seeing things on the internet that pretty much were being justified as art or as being creative and they essentially required skill sets that I already had. so
mg: ok so you are saying like your initial framing you were doing stuff but you hadn't really framed it as being an artwork as such. and then
pp: not quite because I guess I did, I came from, my background originally was in electronic music. so I had a.. as a kid I studied music and I like I did a lot of piano and I played a few instruments and then it's possibly actually my very rigid classical background but then pushed me into more contemporary music or electronic music. Cause it was. It seemed sortof liberating because it was breaking all the rules that I'd had to put up with for very long time. So then I got into electronic music and I did that for almost like 8 or 9 years and went quite deep into that but what's very different from what I was doing then and my practise now is that I distinctly remember having a discussion with someone about how, that I wasn't interested in creating tools, I was really focused on using other people's tools because my importance at that time was to create work and not to create the tools to create work. cause I think there probably a couple of people at that time who were saying, ah you should maybe try your hand at electronics or you know you could make your own software, and I was really not interested at all at that stage.
mg: you wanted to make music not software
pp: yeah and I really felt that spending any time on tools, uhm would totally distract me from my ultimate goal. Even, so I was at the point of really just taking off the shelf software and buying everything that I needed instead of trying to make it myself, whereas today that's kindof like very different. I pretty much make everything myself and I guess a lot of, some of what I am making does consist of tools and I do spend like an enormous amount of time on that.
mg: so you are not making such a clear distinction, you used to make a clear distinction but you are not making such a clear distinction now.
pp: yeah it has become a lot more blurry, but I think
mg: why do you think that happened, do you think .. why do you think that happened?
pp: it's a good question and I am still kind of like asking myself that question because it is true, it is easy to... and I see people who working sortof in my field and working with like openframeworks who have totally lured away by the very honourable task of creating tools and they end up spending most of their time doing that and not actually making work. there is a balance. I guess.
mg: I mean you see you could have stayed and said listen I wanna make work, I don't wanna make tools. because you were a lot clearer about the distinction in the beginning. but then you kindof, it has migrated, I guess you are saying.
pp:I think the big difference is because I sortof have shifted from making electronic music which is essentially largely content driven really, you can get conceptual with electronic music. I am saying music as opposed to sound design or interactive sound installations or experimental sound music

and stuff, it's more music that people would digest I guess, you know, every day as opposed to something more maybe experiential, I dunno. but in that context it kindof does make sense to sortof just focus on the output but if you move into more contemporary art environment which I have kindof done, then it's really about exploring and not being influenced, I think not being influenced by other peoples sortof aesthetic decisions which ?? software is really really critical and if you want to make something that's really original then you really kindof have to do it yourself or at least be really close to that process. I mean there are people who create software with the aid of engineers and assistants and they do so from a distance but I think they do loose certain agency, like you will get a product on the other end, but it will maybe not be as uniquely yours as if you made it yourself.
mg: ok so it's like being engaged in the medium more directly
pp: yeah yeah it's being more heavily engaged in the medium, but the tricky nature this whole sortof code or just technical aspect of this new contemporary art form is that it does restrict the kindof people who are capable of creating it because it's not ..., it doesn't come naturally to everybody, well it doesn't come naturally to anybody, but some people really struggle with this sort of technical skill set so. maybe, maybe that's just a literacy thing that's gonna change in years to come, I am pretty curious to see how that's gonna change.
mg: it's certainly a fascinating field at the moment, it has been fascinating me so yeah
pp: cause a lot people who are working in contemporary art at the moment I feel that some of them are really locked out from some of the really exciting stuff that's happening at the moment simply because they're not technical enough and one of the reactions of the main stream contemporary art world is to sortof label some of this stuff as not art. That's fine and some of it probably isn't art anyway but I think it's a defensive mechanism in the face what is a pretty exclusive activity at the moment.
mg: and your actual creative process, you know like the way that you're, I mean your background is technical so you have lots of technical skills that you are re-using, other people I've spoke to, they have obviously come from different backgrounds and so using software certainly has some technical implications and because you are using the medium it actually changes the way you create?
pp: yeah it does, uhm,
mg: and I also from my own experience if I'm coding stuff which is not for an artwork or if I am coding stuff which is for an artwork it's like ... it's different ... it's almost like they way I am doing it they way my head works or they way I am doing it is different, I don't know if you have experienced anything like that?
pp: to a certain extent I guess you have a lot more liberty when you are making your own art installations, art projects. you are not so bound by,I dunno, rigid requirements and formats and so on. There is definitely a more playful element to it. There's an interesting video by this guy who I really can't remember, but I might send it to you if you haven't seen it. but he talks about, he is interested in, he's really more just interested in tools, and he is interested in increasing the immediacy of coding as a creative process whereby you can, you should be able to understand the correlation between your code and it's output much more clearly and be able to change elements of your code in a much more intuitive way instead of just like typing in numbers to change a variable. Like he has created this tool where you can hover a variable and a slider will come up and you can just without even knowing what variable does you can scrub through a whole bunch of values.
mg:I think there's that quad-v, vvvv, tools which is like a graphical environment,
pp: Yeah I know vvvv
mg: that's similar to that were people can access sliders and actually make things scrub
pp:yeah there was some discussion and comparison between what this guy was doing and the patching environments, there is a somewhat difference because code is typically at a pretty low-level and you really have an enormous amount of freedom to do almost anything you want, whereas in the patching environment you are restricted to these black boxes that you link together and then play around with variables. That's not to demean them as being
mg: it's just different
pp: it is a very different process I think, because I use pure data as a patching tool and I'm familiar with that kind of process, but the code ?? it is a very different one and it is a very powerful but sometimes you can get lost in the implementation details which the attraction of some of this node based stuff. I guess that's where, it's kindof where I am at the moment. This is what really attracted me, for example with openframeworks, where, I was sort of dabbling with c and c++ and playing around with things and creating my own software but nothing that I created really ever a sortof

substantial size in a way in terms of the size of the project because I would reach this level of complexity and then I would get lost in the implementation details. and you know, whatever creative idea was sparked the project initially would fall by the wayside. So all of these creative coding tool-kits have really revolutionised the .. and I think there's more work to be done in terms of improving them and taking them further.
mg: so it's like you are saying there's the process of actually writing the code and there's the conceptual part of it and you've found that the process of actually being involved in the code obscured the conceptual underpinnings.
pp: Yeah it does and it very much can do that. The more lost you get in the complexity of the implementation of the project the easier it is to forget what it was you originally wanted to do.
mg: so in a way to try and keep the complexity in your mind or to focus on the complexity there needs to be ways... no ... to focus on the concept there needs to be ways of handling the complexity of the implementation.
pp: yeah and that happens essentially through methods of abstraction. So the more you can abstract out of the complexity the more exciting, the more you can connect to your actual concepts. That definitely has influenced the way in which I am creating work through code.
mg: I am glad we are having this chat quite late. Because in the mean time I have obviously read a lot about...all sorts of areas and discussed lots of different things with people so, what I am really trying to do is as you are telling me stuff I am kindof trying to rephrase it in terms of the things that I have read without trying to influence your opinion.
pp: that's good
mg: what I am going to do is I am going to let you know, I don't know if there's anything else you want to add, I mean I could ask you about specific engineering processes because one of the things I was looking at, I was looking at software development processes such as Agile and how one could use those kind of processes.
pp: yeah?
mg: To what extent that would work or not work, so that is where my questions started initially. somehow once I started speaking to people they not too worried the actual detail of the implementation problems they are experiencing, what you were just talking about now the whole thing about tools and implementation versus the whole conceptual thing and how that interacts, I find people who are creating artworks, is probably more on people's minds than how the software development process is being controlled.
pp: it's pretty fast and dirty kind of process, which I guess I kind of relish in some way
mg: yeah yeah, the thing is though of course that one could pull some of the engineering processes in but you can't everything in. like for instance I am firm believer of source code control, although it sounds like something too formal it's really awesome from a creative point of view because you can say well what what would it look like if everything was green and you can just branch and make it green.
pp:yeah, no, no I think source control is a very important part of, I think should be a very important part of like a creative coders environment and maybe not everybody, well, people are increasingly using it. but the whole concept of document your work in an art context is hugely important so if the code constitutes your work then source control is a self documenting. It's like an automatic documenting process. when you are branching, when you have got various different ideas and you are branching in various different directions, which you have to when you are exploring a kindof creative space then being to rollback your code to specific point in time and go back down another path is really critical to that process. So yeah I think source control is really important.
mg: and then the other thing which I have come across is this whole concept of testing. If you speak to the engineering people there is quite a bit of focus on testing but I find from an art perspective because you have got the freedom it's maybe not seen as that important, let me not tell you what I think, how do you feel about testing?
pp: I think testing is pretty much ignored, it's done but in very slap dash way in a lot of art installation production, often because there's not a lot of time, because you might be creating something for an event or an installation but it then will be pulled down like and therefore you can't afford to spend enormous amounts of time, I guess it really depends on the longevity of the actual outcome and it's contact with the general public. Like for example if I was to, like I've sort of started thinking about maybe selling one of my pieces if I re-engineered it or if I re-tooled it and that case I would really have to put a lot more work into it and probably maybe even think about some of the more engineering processes that could be involved whereas if I was just exhibiting it then I could .. It's like sortof building a house of cards as long as it doesn't fall down during that time and I would

make sure that it would not do that, but then in internally it could be absolute chaos, a mess. because it is all about the illusion to the general public about what it should be doing or what it is doing. but then yeah if .. and this is sort of one of the things one of the kind of teething issues that I guess if people working with code wanna to eventually consider themselves artist in a more mainstream commercial way in the sense being able to create works that are collectible this problematic because I am not even sure if in the future the art market is going to remain anything like it looks today, I mean this whole concept of collecting pieces of visual art may, not disappear, but it may be completely marginalised in the face of something new that will emerge that we haven't really seen yet. but if, let's say for the sake of argument that the art market will continue and then in order to be recognised as artists, code work creators would have to become part of that, then I think the software produced for the general public will have to be more robust and that will definitely require more engineering process and more testing.
mg: but there is a certain transience to the medium, to the software medium ...
pp: yeah there certainly is and that's why the source control is really critical too, it's creating a snap shot at a particular time that can't always be re-created at a later stage. It gives you a sortof point to be able to, you know, reformulated it using an updated platform or whatever in the future. That's another reason why I like source control is really really critical because a lot of the stuff is incredibly impermanent like even in the space of three months a piece of software can become out of date because it can't run on the latest platform.
mg: or just that the specific installation is like that and works like that, even an installation in a space is there for three weeks or whatever and then it's gone and the immersion or the interaction is never like that again so I guess a part of it is that it's a transient medium.
pp: it might also be a very transient medium at the moment because it's very emerging too. you know, if it were to become a lot more, if this sort of thing were to become a lot more mainstream then maybe that transience would sortof, not disappear but it would be reduced somewhat. I guess people are not afraid to throw away a lot of stuff or to just let things.
mg: from an artist's point of view now, I'm asking, are you saying that artists aren't afraid to throw away.
pp: Uhm yeah well I think it's important, in any creative field, I think it is important to be able to know what to throw away and what to keep, like that's a very critical and part of refining a work is like knowing when it's close to completion and knowing what to get rid of and to cull in order to achieve.
mg: so that's almost like a testing in a way...?
pp: yeah
mg: ... so it's like you have these ideas conceptually or whatever and you have to decide this will fly this won't fly?
pp: uhm but there's another aspect to this I think maybe a lot of people don't really know the value of what they are creating now and maybe people are just throwing away code because they can there's no reason not to. or maybe the medium is inherently impermanent ....
mg: ... evolving
mg: ok I think I have got enough to chew on, what I am going to do is I'll transcribe this interview and then I'll send you a copy of it.
pp:ok
mg:if that's ok with you and what I'll also do is once I've written everything up and it is all in with the supervisors and everything and I am done I'll send you a copy of what it looks like in my thesis.
pp: yeah that will be great, so are you doing this for a masters or an honours
mg: yeah it's a master's degree at the university of the Witwatersrand, it's a university in Johannesburg. and it;s in their digital arts department but I got co-supervision from the arts department and from the engineering department so it has been an interesting process.
pp: how has your engagement with the engineering department been, what do they think
mg: well I have this supervisor, there are some parts of what I am doing he is completely flabbergasted doesn't even know what to say, he can tell me things about the software processes, and that's interesting, but I presented it to the engineering students at their post-graduate symposium, it's like they had this confused look on their face and they couldn't really even understand why someone would want to, why someone was even wondering about these things in the first place.
pp: yeah I think this is one of the reasons, when I started getting into more art stuff, I was kind of surprised because I never expected myself to be in this position, I just never saw myself as an artist. so I sortof felt like I needed to justify myself somehow and I kindof looked back and tried to think

of any moments that might have demonstrated an interest in what I was doing now today and there was one moment like when I was studying undergraduate engineering in Melbourne in Australia and I just remember everybody essentially the bulk of the people in the lecture theatre were essentially just sortof I guess, cause I studies arts and engineering, I did a double degree It was liberal arts, not fine arts, and I kindof experienced it as schizophrenia when I would go from one class to another because the art students in my literature classes would all be about critical thought and then the engineering classes it was all about just absorbing the information just sucking it down and not questioning and the sort of rote learning and the sort of figuring out the problems and so on.
mg: and how to make it be faster and the teams more efficient and less bugs
pp: yeah it was all about efficiency and speed but then nowhere during that entire period did anybody sortof ask any questions about why are we making this faster or why are we making this robust. none of these philosophical questions were ever raised and this really bothered me because if I am to throw myself at anything and be passionate about something I kindof want to know why I'm doing, there was none of this. This was I guess what really defined me.
mg: It's really a conceptual thing vs the process thing seems to be a recurring theme for me on this project. That's what it is the concepts around it are like why are we doing this or how does it fit in or that questioning aspect which I think to a certain extent could relate to the conceptual elements of making a digital artwork.
pp: The sad thing is, I actually think it is really important to even you know the general regular engineering as well, I think uhm this lateral thinking and this questioning can be useful in terms of creating more innovative solutions.
mg: yeah yeah
pp: people suffer from just accepting black as black and white as white.
mg: or that's the fastest solution with the resources we have that's the best, quickest way to do it.
mg: ok great. At the moment I think that's enough for now.
pp: ok cool
mg: I'll give you, I'll send you, if you want I can send you the audio file of the conversation.
pp: no that's fine, if you have a transcript.
mg: I have to transcribe it to put in my document so I'll do a transcript and then send the final document to you as well and then you can see, hopefully I don't misquote you.
pp:yeah I'd be interested to see.
mg: Ok thank you so much, I appreciate you staying up late.
pp: no that's fine.
mg: so we'll be in touch.
pp: ok thanks again bye bye
mg: thanks ok bye

Appendix VI: Email interview with Nathaniel Stern

Nathaniel Stern nathaniel.stern@gmail.com
11/20/11

to me
Hi Maia:

I think as with any tool, of course, the possibilities and attributes effect/affect your thinking and outcome. This can be seen in the difference in my and others' work, for example, when they have gone from something like Director to Max or Max to OpenFrameworks. Then, the more one learns about varying tools, the more their options fan out, and they can start thinking across various modes of making, then worry about the language/code/software to use later.

One researcher who has done a bit more thinking than I have around code and making is Pall Thayer. I'd recommend searching for "pall thayer code" in google and starting there for your research. Good Luck! Best,

nathaniel
http://nathanielstern.com

On Nov 19, 2011, at 4:53 PM, maia grotepass wrote:

>
> Hi Nathaniel
>
>
> I got your contact info from my supervisor, Tegan Bristow. She said
> you may be able to help me. I am busy with my M.A at the
> Digital Arts division at the University of Witwatersrand in
> South Africa. I am looking for artists who use software/code
> as art medium. This is for the written research part of my
> M.A. I would like to get some opinions from artists about how
> they think using software has affected their creative
> practise.
>
> The questions are:
>
> Has using software as medium changed your creative process? If so how?
> What problems or benefits do you experience when creating an artwork using software?
>
> I would appreciate any views you have. Also if you know of
> anyone who would be willing to discuss these questions with
> me, I would really appreciate it if you would pass on this
> email or send me their details and I will contact them
> directly.
> Hoping to hear from you.
> Maia Grotepass
>
> www.maiatoday.co.za

Appendix VII: Email interview with Pall Thayer

Pall Thayer pallthay@gmail.com
Jan 14
to me
Hi Maia,

Of course, I'm always happy to hear from people who are interested in my work. I certainly hope that there are others out there who can, in some way, connect with it.

I think that to understand why I'm creating these "Microcodes" you kind of have to know how I got there. It's a complicated and long story but I'll try to make it short and simple.

I began using computers when some of the very first "home" computers began to emerge in the late 70's - early 80's. My family bought an Atari 400 computer in 1980 and at the same time, Apple II computers started appearing at my school. I was intrigued enough to learn some Basic (the programming language) programming so that I could explore the possibilities of these machines on my own.

When I began my own serious art studies, I was a painter and drawer. At the time, I wasn't really thinking about computers in artistic terms although I was still using them and writing programs on them. While I was doing my Bachelor's studies at the Art Academy in Iceland, I went as an exchange student to the Helsinki Academy of Arts as an exchange student in their "Time and Space" department. The program's facilities consisted almost entirely of video editing equipment and computers. While there, I was introduced to some software that allowed for interactive audio-visual design and I began reconsidering my painting practice within this realm of what I could do with computers and interaction.

Here is a key point: Within my painting practice, I had been exploring abstraction from an approach that involved the artist creating a conceptual atmosphere and then separating himself from the creative act, allowing interactive elements to take over and create the actual work. The internet represented to me an ideal vehicle for this approach. Through programming, I could create systems that a multitude of people could interact with and their interaction would create the actual work. As I explored the possibilities that this provided, the programming code became more and more important as the creative conduit in these pieces. I began to consider the idea that the art was not what was being created by users' interactions, it was the code itself.

So, in direct response to your question, I would not say that USING software as a medium has changed my creative process. Rather that CREATING software as a medium has changed my creative process.

I'll tell you the story of what prompted me to start creating my Microcodes. A few years ago, I created an online piece that attracted some attention. It was titled "On Everything". It was based on software that I created that systematically traversed the entire collection of photographs on Flickr.com in the order that the photos

were uploaded. It would take each of these images and "reinterpret" them as "paintings", painting each one to the screen and then the next one would be painted over the last. This would eventually result in a complex collage of unrelated imagery that was constantly being regenerated over time. While this was going on, a very mechanical, computerized voice would read recent blog posts that were being culled from Blogger.com, providing a suggestive narrative that actually had nothing to do with the imagery being "painted" to the screen.

As I mentioned, this piece garnered some attention and some articles were written about it. However, the articles I saw suggested that the work selected random images from Flickr.com. I was surprised because I had released the source code for the work and it was quite obvious, if one read the code, that it was not random. It was systematic. To me, this was an important aspect of the work. But it was obvious that no one was interested enough in the code to actually read it. So I began to wonder, what would it take to make people acknowledge the programming code? As far as I was concerned, that was my "creative intervention". The images were not mine, the text was not mine, but what my code did to these elements made them mine.

My personal answer to the question, "What would it take to make people acknowledge the code?" was that I would have to create work that was only code. Do away with the audio-visual elements and only display the code. The work that I've created based on this idea has garnered quite a bit of attention but it's been limited to very specific groups. Obviously, groups that have some sort of understanding of code. Many of the Microcodes don't actually produce anything on-screen. Their significance is embedded in a reading of the code itself. They are all executable and will run without producing errors but what they're doing isn't obvious unless the viewer is able to understand the meaning of the programming code. In some ways, it crosses into the realm of poetry.

But, since you contacted me with your question, I think I should point out a small flaw that I see in it. I, and many others working in similar ways, don't USE software. We CREATE software. There is a big difference there.

The way I see it. My primary medium is computer programming code. But this can be problematic because it can create a large divide between my medium and the viewer's medium. In cases where I write software that does actually produce audio-visual effects on the screen, my medium is the code but the viewer's medium is the screen. They don't see the medium that I used to create the end product. In painting, the viewer is looking at the artist's medium. This is not the case with software art. This is why I think it's important to make people aware of the code. Force them to acknowledge it and, perhaps, to try to understand it as well.

I hope this helps with your research. Feel free to contact me again if you have any further questions or would like any further input.

Best regards,
Pall Thayer
On Wed, Jan 11, 2012 at 5:45 AM, maia grotepass <maiatoday@gmail.com> wrote:
> Hi Pall
>
> Sorry for this cold email. I was looking at our work on the internet,

> in particular the microcodes. I find them very witty and succinct.
> Little puzzles on more than one level: I like getting them running and
> seeing the results and then looking at the code to see what you did
> and then, what really makes it interesting, is figuring out the art
> references.
>
> I am a student at the Digital Arts division at the University of
> Witwatersrand in Johannesburg, South Africa. I am in the process of
> completing my M.A in digital art. I am looking for artists who use
> software/code as art medium. This is for the written research part of
> my M.A. I would like to get some opinions from artists about how they
> think using software has affected their creative practise. Your works
> interest me particularly because you present code snippets. I find
> this interesting because often artists use code but do not see the
> code as part of the end product, but rather just a tool to get an
> effect.
>
> The questions are:
> Has using software as medium changed your creative process? If so how?
> What problems or benefits do you experience when creating an artwork
> using software?
>
> I would appreciate any views you have. No need to spend a lot of time
> if you are busy. I have read what you have written in the pdfs on your
> website too. If writing something is too time consuming we can also
> have a chat on skype which I could transcribe. Whatever is easiest or
> you.
>
> Hoping to hear from you.
>
> Maia Grotepass
> www.maiatoday.co.za
--
****
Pall Thayer
artist
http://pallthayer.dyndns.org

9 List of Works Cited

Abrahamsson, Pekka et al. *Agile Software Development Methods*. Vol. 478. Citeseer, 2002. Web. 9 Mar. 2012.

Alexander, Amy. “SVEN | Amy Alexander.” *amy-alexander.com*. Web. 8 Mar. 2012.

Amabile, Teresa. “How to Kill Creativity.” *The Hudson Review* 57.3 (2004): 515.

Bach, James. “Exploratory Testing Explained.” *Interpreting* (2003): 1–10.

---. “What Is Exploratory Testing?” *Stickyminds.com*. 2001. Web. 19 Feb. 2012.

Beck, Kent. “Embracing Change with Extreme Programming.” *Computer* 32.10 (1999): 70–77. Web. 12 Feb. 2012.

Beck, Kent et al. “Manifesto for Agile Software Development.” *www.agilemanifesto.org*. 2001. Web. 11 Apr. 2010.

Biggs, Simon. “Make or Break? Concerning the Value of Redundancy as Creative Strategy.” 2011.

Blais, Joline, and Jon Ippolito. *At the Edge of Art*. Thames & Hudson, 2006.

Boehm, Barry W., and Richard Turner. *Balancing Agility and Discipline: a Guide for the Perplexed*. Addison-Wesley, 2003. Web. 1 Oct. 2010.

Bunt, Brogan. “About | Brogan Bunt.” 2011. Web. 4 Mar. 2012.

---. “Computational Drawing : Code and Invisible Operation.” *ISEA 2011 Instanbul*. Istanbul, 2011. Print.

---. “Risking Code: Software Art-dilemmas and Possibilities.” *University of Wollongong Thesis Collection* 2007 : 1–134. Web. 9 Mar. 2012.

Buss, Keno. “Behavioural Patterns for the Analysis of Creative Behaviour.” *Analysis* 2011 : 1–269.

Candy, Linda, and Ernest A. Edmonds. “Modeling Co-creativity in Art and Technology.” *Proceedings of the 4th Conference on Creativity and Cognition*. Loughborough, UK: ACM, 2002. 134–141. Web. 11 Apr. 2010.

Chun, W.H.K. *Programmed Visions: Software and Memory*. The MIT Press, 2011. Print.

Cramer, Florian. “Concepts, Notations, Software, Art.” *Seminar for Allegmeine Und Vergleischende Literaturwissenschaft*. 2002. 1–11. Web. 25 Feb. 2012.

Cramer, Florian, and Matthew Fuller. “Interface.” *Software Studies: a Lexicon*. Ed. Matthew Fuller. MIT Press, 2008. 149–152. Print.

Csikszenthihalyi, Mihaly. “Implications of a Systems Perspective for the Study of Creativity.” *Handbook of Creativity*. 1st ed. Ed. Robert J. Sternberg. Cambridge, England: Cambridge University Press, 1998. 313–335. Print.

Dietrich, Arne. “Who ’ s Afraid of a Cognitive Neuroscience of Creativity ?” *Creativity Research Journal* 42 (2007): 22–27.

Edmonds, Ernest A. et al. “The Studio as Laboratory: Combining Creative Practice and Digital Technology Research.” *International Journal of Human-Computer Studies* 63.4-5 (2005): 452–481. Web. 11 Apr. 2010.

Edmonds, Ernest A., and Linda Candy. “Creativity, Art Practice, and Knowledge.” *Communications of the ACM* 45.10 (2002): 91–95. Web. 5 Aug. 2010.

“File:Windows XP Blue Screen of Death (PAGE FAULT IN NONPAGED AREA).svg - Wikipedia, the Free Encyclopedia.” *en.wikipedia.org*. Web. 12 Mar. 2012.

Finke, Ronald A, T. B Ward, and Steven M Smith. *Creative Cognition: Theory, Research, and Applications*. Cambridge, Mass: M.I.T. Press, 1992. Print.

Fiore, Q., and Marshall McLuhan. *The Medium Is the Massage*. Bantam Books, 1967. Print.

Fischer, Gerhard. “Social Creativity: Turning Barriers into Opportunities for Collaborative Design.” *Proceedings of the Eighth Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and practices-Volume 1*. ACM, 2004. 152–161. Web. 4 Mar. 2012.

Fowler, Martin, and Kent Beck. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. Web. 8 Nov. 2010.

Galloway, Alexander. “Alexander R. Galloway’s Home Page.” Web. 24 Feb. 2012.

George, James, Greg Borenstein, and James Hughes. “ofxAddons.” *ofxaddons.com/*. 2012. Web. 9 Feb. 2012.

“GitHub.” *github.com*. Web. 5 Mar. 2012.

Goldberg, Joshua. “Dervish.” 2007. Web. 2 Aug. 2012.

---. “Dervish-meditation 2012 on Vimeo.” *http://vimeo.com/*. 2012. Web. 2 Aug. 2012.

---. “Joshuagoldberg.” *goldbergs.com*. Web. 7 Mar. 2012.

Grotepass, Maia. “1 Nullpointerexception-commentcompile.” *vimeo.com*. 2012. Web. 2 Mar. 2012.

---. “2 Nullpointerexception-commitoften.” *vimeo.com*. 2012. Web. 2 Mar. 2012.

---. “3 Nullpointerexception-initbefore.” *vimeo.com*. 2012. Web. 2 Mar. 2012.

---. “4 Nullpointerexception-interfaceinstead.” *vimeo.com*. 2012. Web. 2 Mar. 2012.

---. “Maiatoday.” *maiatoday.blogspot.com/*. Web. 1 Oct. 2009.

---. “Maiatoday (maiatoday).” *github.com*. 2012. Web. 5 Mar. 2012.

Guilford, JP. “Creativity Research: Past, Present and Future.” *Frontiers of Creativity Research*. Ed. Scot G Isaksen. Buffalo: Bearly Ltd, 1987. 33–65. Print.

“Has Being an Artist Changed the Way You Code or Vice Versa - openFrameworks Forum.” *forum.openframeworks.cc*. 2011.

Hausman, C.R. *A Discourse on Novelty and Creation*. State Univ of New York Pr, 1975. Web. 16 Feb. 2012.

Hayles, N. Katherine. *My Mother Was a Computer: Digital Subjects and Literary Texts*. 1st ed. Chicago: University Of Chicago Press, 2005. Print.

---. “Print Is Flat, Code Is Deep: The Importance of Media-Specific Analysis.” *Poetics Today* 25.1 (2004): 67–90.

---. “Traumas of Code.” *Critical Inquiry* 33.1 (2006): 136–157. Web. 21 Sept. 2009.

Highsmith, Jim, and Alistair Cockburn. “Agile Software Development: The Business of Innovation.” *Computer* 34.9 (2001): 120–127. Web. 15 Jan. 2012.

Ippolito, Jon, and John Simon. “From the Rhizome Archives: Code As Creative Writing--An Interview with John F. Simon, Jr. by Jon Ippolito.” *rhizome.org*. 2003. Web. 10 Apr. 2011.

Itkonen, Juha, and Kristian Rautiainen. “Exploratory Testing: a Multiple Case Study.” *Empirical Software Engineering, 2005. 2005 International Symposium On*. Vol. 00. IEEE, 2005. 10–pp. Web. 19 Feb. 2012.

Kaner, Cem. “Exploratory Testing.” *Science* 2006.

Kirsner, Scott. “Wired 14.03: The Chaos of Joshua Davis.” *WIRED*. 2006. Web. 24 Feb. 2012.

Kittler, Friedrich. “There Is No Software.” *CTheory. net* (1995): 1–7. Web. 21 Jan. 2012.

Krysa, Joasia, and Grzesiek Sedek. “Source Code.” *Software Studies: a Lexicon*. Ed. Matthew Fuller. MIT PRess, 2008. 236–242. Print.

LeWitt, S. “Oral History Interview with Sol LeWitt, 1974 July 15 - Oral Histories | Archives of American Art, Smithsonian Institution.” *Smithsonian Institution*. 2007. Web. 26 Jan. 2012.

---. “Paragraphs on Conceptual Art.” *Artforum* 5.10 (1967): 79–83. Web. 4 Jan. 2012.

Lee, John. “Goodmans Aesthetics and the Languages of Computing.” *Aesthetic Computing*. Ed. Paul Fishwick. MIT Press, 2006. 29–42. Print.

Lovejoy, Margot. *Digital Currents: Art in the Electronic Age*. 3rd ed. New York: Routledge, 2004.

Lubart, Todd I. “Models of the Creative Process: Past, Present and Future.” *Creativity Research Journal* 13.3 (2001): 295. Web. 7 June 2010.

Mahoney, M.S. “The History of Computing in the History of Technology.” *Annals of the History of Computing* 10.2 (1988): 113–125. Web. 24 Feb. 2012.

Malmberg, Elise. “Apple - Pro - Profiles - Joshua Davis.” Web. 18 June 2010.

Manovich, Lev. “New Media from Borges to HTML.” *The new media reader* (2002): n. pag.

---. *The Language of New Media*. Cambridge, Mass: MIT Press, 2001. Print.

Marchese, F. T. “The Making of Trigger and the Agile Engineering of Artist-Scientist Collaboration.” *Information Visualization, 2006. IV 2006. Tenth International Conference On*. IEEE, 2006. 839–844. Web. 24 Oct. 2010.

McLuhan, Marshall. *Understanding Media: The Extensions of Man : Critical Edition*. Critical. Gingko Press, 2003.

“Medium - Definition of Medium - Synonyms, Pronunciation, Spelling from Free Dictionary.” *www.thefreedictionary.com*. Web. 4 Mar. 2012.

“Metaphor - Definition of Metaphor by the Free Online Dictionary, Thesaurus and Encyclopedia.” *www.thefreedictionary.com*. Web. 5 Mar. 2012.

“MoMA | The Collection | Kazimir Malevich. Suprematist Composition: White on White. 1918.” Web. 4 Aug. 2012.

Myers, Glenford J et al. *The Art of Software Testing , Second Edition*. 2nd ed. Hoboken, New Jersey: John Wiley & Sons, Inc.,, 2004. Print.

“National Gallery of Iceland.” Web. 5 Aug. 2012.

Novikov, Michael, and Nicolas Heuser. “Agile Software Development.” 2006. Web. 7 Jan. 2012.

Plotkin, Andrew. “Boodler Home.” *boodler.org*. 2009. Web. 9 Mar. 2012.

Pope, Rob. *Creativity: Theory, History, Practice*. London: Routledge, 2005. Print.

Proske, Pierre. “About « Digitalstar.net.” *www.digitalstar.net*. Web. 7 Mar. 2012.

---. “Abstract Microecologies.” 2006. Web. 2 Aug. 2012.

“Rhizome | Do Artists and Technologists Create Things the Same Way? Seven on Seven Guests Respond.” *rhizome.org*. 2011. Web. 10 June 2011.

Riehle, Dirk. “A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming : How Methodolo- Gies May Learn from Each Other.” *Review Literature And Arts Of The Americas* Xp 2000 (2000): 1–13. Print.

Runco, Mark, and Ivonne Chand. “Cognition and Creativity.” *Educational psychology review* 7.3 (1995): 243–267. Web. 17 Feb. 2012.

“Seven on Seven - Rhizome.” *rhizome.org*. Web. 26 Feb. 2012.

Shanken, EA. “Art in the Information Age: Technology and Conceptual Art.” *Leonardo* 35.4 (2002): 433–438. Web. 3 Feb. 2012.

Shneiderman, Ben. “Creativity Support Tools: Accelerating Discovery and Innovation.” *Communications of the ACM* 50.12 (2007): 20–32. Web. 4 Mar. 2012.

Shore, James, and Shane Warden. “The Art of Agile Development.” (2007): 409. Web. 15 Aug. 2010.

“Small Sailors’ Home - Kurt Schwitters - WikiPaintings.org.” Web. 4 Aug. 2012.

Smith, Jeff, David Mould, and Mark Daley. “Constructures: Supporting Human Ingenuity in Software.” *Digital Creativity* 20.1 (2009): 79–94. Web. 21 May 2011.

Stern, Nathaniel. “Nathaniel Stern : Given Time, Networked Installation and Continuous Performance, February 2010.” Web. 9 Aug. 2012.

---. “Nathaniel Stern : Artist Statement and General Interests.” *http://nathanielstern.com/artistic-inquiry-artist-statement/*. Web. 26 Feb. 2012.

---. “Nathaniel Stern : Short Artist Biography.” *http://nathanielstern.com/about/*. Web. 26 Feb. 2012.

---. “Nathaniel Stern : Stuttering, Interactive Installation, 2003.” Web. 5 Aug. 2012.

Sternberg, R J. “What Is the Common Thread of Creativity? Its Dialectical Relation to Intelligence and Wisdom.” *The American psychologist* 56.4 (2001): 360–2.

Thayer, Pall. “Microcodes - Pall Thayer.” *pallit.lhi.is/microcodes/*. Web. 24 Feb. 2012.

---. “On Artists That Write Code and Artists That Do Not.” 2008 : 94105–94105. Print.

---. “Pall Thayer - Artist.” *pallthayer.dyndns.org/*. Web. 24 Feb. 2012.

---. “The Microcode Primer A Guide for Non-coders Towards a Conceptual Appreciation of Code.” *pallit.lhi.is/microcodes/*.

---. “Pall Thayer Bio and Interview.” Web. 1 Mar. 2012.

Tribe, Mark. “BIO + CV - Mark Tribe’s Portfolio.” Web. 24 Feb. 2012.

Tribe, Mark, and Reena Jana. “New Media Art - Mark Tribe - Brown University Wiki.” Web. 18 June 2010.

Trifonova, A., L. Jaccheri, and K. Bergaust. “Software Engineering Issues in Interactive Installation Art.” *International Journal of Arts and Technology* 1.1 (2008): 43–65. Print.

Wands, Bruce. *Art of the Digital Age*. New York: Thames & Hudson, 2006. Print.

Weisberg, Robert W. *Creativity: Understanding Innovation in Problem Solving, Science, Invention, and the Arts*. Hoboken, N.J: John Wiley & Sons, 2006. Print.

Whitelaw, Mitchell, and Paul Prudence. “(the Teeming Void): An Interview with Paul Prudence (for Neural 40).” *teemingvoid.blogspot.com*. Web. 19 Jan. 2012.

Wright, Richard. “Data Visualization.” *Software Studies: a Lexicon*. Ed. Matthew Fuller. MIT Press, 2008. 78–86. Print.

“openFrameworks.” *www.openframeworks.cc*. Web. 1 Oct. 2009.