



UNIVERSITY OF TWENTE.

Creative Technology
EEMCS faculty

Software Design within Creative Technology

Written by
Hannah Ottenschot

02-07-2021

Supervisors
Ansgar Fehnker
Champika Epa Ranasinghe



Abstract

The presence of software design with the study Creative Technology is examined. An opportunity for the curriculum is identified, where the addition of software design could benefit students following the program. By conducting interviews with alumni and teachers and hosting a survey for all Creative Technology students and alumni, common opinions and possible solutions are extracted. These findings are summarized in chapters 4 and 5. Iteratively, a design is created for a software design workshop. The workshop takes two days and takes place in the third module of Creative technology. The manual for the workshop is described in chapter 6. Due to the impossibility of testing the workshop in its intended environment, an evaluation workshop is set up based on the ideas for the original software design workshop. Additionally, Teaching Assistants and teachers give feedback on the design during evaluation sessions. These sessions are described in chapters 7 and 8. The final design for the software design workshop is described in chapter 9. In chapter 10, the design itself and the process of creating the design are reflected upon and in chapter 11 future research directions are presented.

Acknowledgements

I would like to thank Ansgar Fehnker and Champika Epa Ranasinghe for their time, help, and ideas throughout the graduation semester. Additionally, I would like to thank all participants of the interviews, survey, and evaluation sessions for their time and feedback. Without their insightful comments and willingness to think along, this project would not have been successful. Finally, I would like to thank Sebastiaan van Loon and my family for their emotional support and willingness to provide a listening ear.

Table of contents

List of Figures	6
Introduction	9
Background Research	11
2.1 Current state of Computer Science curricula	11
2.1.1 Novice programmers	11
2.1.2 Teaching Computer Science	12
2.2 Methods for teaching Software Design to novice students	13
2.2.1 Difficulties	13
2.2.2 Solutions and their benefits	14
2.2.3 Discussion	15
2.3 Informal learning	16
2.3.1 (College) Hackathon	16
2.3.2 Makeathon	17
Methods and techniques	18
3.1 Iterative process	18
3.2 Outline thesis	19
Iteration 1: Design space exploration	20
4.1 Stakeholders	20
4.2 Interviews	21
4.2.1 Interviews with teaching staff	21
4.2.2 Interviews with alumni	23
4.2.3 Elicited requirements	25
4.2.4 Project outline	25
4.2.5 Project content	26
4.2.6 Design	33
Iteration 2: Student requirements	34
5.1 Survey	34
5.1.1 Survey results	34
5.1.2 Updated list of requirements	40
Iteration 3: Workshop manual	41
6.1 Brainstorm with teaching staff	41
6.2 Final version of the introductory workshop manual	41
Iteration 4: Evaluation workshops	42
7.1 Evaluation workshop	42

7.1.1 Layout	42
7.1.2 Content	43
7.1.3 Justification	44
7.1.4 Results	45
7.1.4.1 Pilot session	45
7.1.4.2 Evaluation workshop results	46
Iteration 5: TA validation	50
8.1 Evaluation with TAs	50
Iteration 6: Final prototype	53
9.1 Software Design workshop	53
9.1.1 Layout software design workshop	53
9.1.2 Content software design workshop	53
9.1.2.1 Lecture 1.1: Introduction to software design	53
9.1.2.2 Lecture 1.2: Explanation of assignment 1	54
9.1.2.3 Assignment 1	54
9.1.2.4 Lecture 2.1: Explanation of assignments 2 and 3	54
9.1.2.5 Lecture 2.2: Good coding habits and common mistakes	54
9.1.2.6 Live coding session	54
9.1.2.7 Lecture 2.3: Introduction to Git	55
9.1.2.8 Assignment 2	55
9.1.2.9 Assignment 3	55
9.1.3 Grading	56
Conclusion	57
10.1 Research questions	57
10.2 Reflection on design process	59
Future Work	60
11.1 Workshop evaluation	60
11.2 Workshop	60
11.3 Involvement programme staff	61
11.4 Connection to other modules	61
Appendix 1: Interview summaries	62
Appendix 2: First draft introductory workshop manual	65
Appendix 3: Student survey summary	66
Appendix 4: Final version of the introductory workshop manual	81
Appendix 5: Evaluation workshop content	98
5.1: What is software design?	98
5.2: Assignment 1	101

5.3: Explanation of original workshop	105
5.4: Good coding habits	107
5.5: Assignment 2	110
Appendix 6: Evaluation workshop results	116
1: Pilot session	116
2: Workshop 1	117
3: Workshop 2	120
4: Workshop 3	127
Appendix 7: TA evaluation presentation slides	133
Appendix 8: Workshop content	137
1: What is software design?	137
2: Assignment 1 introduction	148
3: Assignment 1	152
4: Assignment 2 & 3 introduction	153
5: Good habits and common programming mistakes	156
6: Introduction to Git	159
7: Assignment 2	166
8: Assignment 3	166
References	167

List of Figures

Figure number	Description	Page number
1	Iterative design process used for this project	19
2	Positive sides and obstacles of software design and general remarks mentioned during teaching staff interviews	21
3	A summarized list of things alumni mentioned they thought should be kept within CreaTe	22
4	A summarized list of things alumni mentioned they thought should be added to CreaTe	23
5	Positive sides and obstacles of software design and general remarks teaching staff mentioned	24
6	Activity idea #1: guest lecture, with pros and cons	25
7	Activity idea #2a: Add software design to existing programming courses, with pros and cons	26
8	Activity idea #2b: Add software design to existing design courses, with pros and cons	27
9	Activity idea #3a: One-time workshop(s), with pros and cons	28
10	Activity idea #3b: Spread out workshop(s), with pros and cons	29
11	Activity idea #3c: One-time or spread out workshop(s), with pros and cons	30
12	Activity idea #3c: One-time or spread out workshop(s), with pros and cons	31
Appendices		
13 - 19	Slides of the “What is software design?” presentation for the evaluation workshop	98 -101
20 - 27	Slides of the “Assignment 1” presentation for the evaluation workshop	101 - 105
28 - 30	Slides of the “Explanation of original workshop” presentation for the evaluation workshop	105 - 106
31 - 36	Slides of the “Good coding habits” presentation for the evaluation workshop	107 - 108

37 - 38	Slides of the “Assignment 2” presentation for the evaluation workshop	110
39	Output of the code given to evaluation workshop participants	111
40	Class diagram made during the evaluation workshop pilot session	116
41	Interaction diagram made during the evaluation workshop pilot session	116
42	Use case diagram made during the evaluation workshop pilot session	117
43	Class diagram made by participant 1 during evaluation workshop 1	117
44	Class diagram made by participant 2 during evaluation workshop 1	118
45	Interaction diagram made by participant 2 during evaluation workshop 1	118
46	Use case diagram made by participant 2 during evaluation workshop 1	119
47	Class diagram made by participant 1 during evaluation workshop 2	120
48	Class diagram made by participant 2 during evaluation workshop 2	120
49	Interaction diagram made by participant 2 during evaluation workshop 2	121
50	Class diagram made by participant 3 during evaluation workshop 2	121
51	Interaction diagram made by participant 3 during evaluation workshop 2	122
52	Use case diagram made by participant 3 during evaluation workshop 2	122
53	Class diagram made by participant 1 during evaluation workshop 3	127
54	Interaction diagram made by participant 1 during evaluation workshop 3	128
55	Use case diagram made by participant 1 during evaluation workshop 3	128
56	Class diagram made by participant 2 during evaluation workshop 3	129
57	Interaction diagram made by participant 2 during evaluation workshop 3	130
58	Use case diagram made by participant 2 during evaluation workshop 3	131
59 - 66	Slides of the TA evaluation presentation	133 - 136
67 - 79	Slides of the “What is software design?” presentation	137 -143

80	Class diagram example. [50]	145
81	Interaction diagram example. [50]	146
82	Use case diagram example. [50]	147
83 - 90	Slides of the “Assignment 1” presentation	148 - 151
91 - 96	Slides of the “Assignment 2 & 3” presentation	153 - 155
97 - 102	Slides of the “Good coding habits and common programming mistakes” presentation	156 - 158
103	Location of the “Sign in” button for the Git introduction	159
104	Location of the “Projects” button for the Git introduction	160
105	Location of the “Your projects” button for the Git introduction	160
106	Location of the “New project” button for the Git introduction	160
107	Location of the the “Project name” field and the “Create project” button for the Git introduction	161
108	SNT’s Command line instructions for the Git introduction	162
109	Invite member section for the Git introduction	165

1. Introduction

Creative Technology (CreaTe) is a study at the University of Twente. It focuses on multiple disciplines, including computer science, electrical engineering, and design. Students each have their background, some more technical than others. As a study that is internationally focussed and has different degree-specific requirements than most University studies, students with many different nationalities and backgrounds join the program. The focus is on using the latest technologies for creating solutions that have an impact on society. [26]

Within CreaTe, it is not a rare sight that students dismiss programming as something they are just unable to do. And such a comment is not specific to CreaTe: learning how to program is found to be challenging by many [1, 28]. Over the years, different University studies that include programming courses have been tackling this problem. Many of these programs grade their students mainly according to whether the programs they write produce valid output. There is less focus on understanding programming concepts and using maintainable structures. However, studies argue that this should change. Instead, programming should be treated more like a problem-solving activity: first, the problem needs to be determined. Then, a plan should be made on how to tackle this problem, followed by the creation of the solution. This step-by-step approach can help students understand what they are doing more clearly, making it easier for them to explain to a teacher what they need help with. In short, programming courses should focus more on software design [1, 2, 3].

However, what exactly is software design, and what does it entail? In a general sense, software design can be seen as the process of designing a piece of software based on requirements, interviews with stakeholders, and (technical) constraints [27]. How exactly this process is executed varies. For this thesis, software design will be treated as a very general term. All the activities students use to prepare for writing a piece of code, formal and informal, and the creation of diagrams to show what a piece of code entails after writing are considered part of the software designing process.

This thesis focuses on the CreaTe curriculum and the possibilities and effects of including more software design topics. It should be noted that changing a curriculum does not come without consequences. Especially for a broad study such as CreaTe, it is hard to determine what is best to put in a curriculum. Since so many topics need to be addressed within a limited amount of time, choices must be made. To determine the possibilities, different aspects of the study need to be examined. To get an insight into what involved parties think of this, students, teaching staff, and alumni are interviewed. Several research questions guide the gathering of information:

- **RQ 1:** In which parts of the CreaTe curriculum is software design currently present?
 - **RQ 1.1:** Do lecturers think the current teachings on software design are adequate?
 - **RQ 1.2:** What are students' experiences with software design within CreaTe?
 - **RQ 1.3:** Do alumni feel as if the teachings on software design were good enough to work within a programming-related field?

Once it is known where within CreaTe software design is already present and the opinions of the parties involved have been determined, possible places for (more) software design related topics

are explored. Next to that, it is explored how other institutes tackle teaching software design to students, which teaching methods are being used, and why. Important aspects of existing methods for teaching software design to students are extracted. Based on the interviews and research, the possible shapes software design could take within the CreaTe curriculum are discussed. The following research questions guide this process:

- **RQ 2:** How can the CreaTe curriculum best be changed regarding Software Design?
 - **Sub-RQ 2.1:** Where within the curriculum would material on Software Design fit best?
 - **Sub-RQ 2.2:** Which aspects of Software Design are best suited for CreaTe?

The goal is to, if possible, create an educational solution that is also fun for students. This aim connects to the goals and existing courses of CreaTe, and it is expected that such a solution will be accepted more easily by students.

2. Background Research

Before the ideation part of this thesis can start, it is important to examine existing literature on the topic. While the focus of this thesis is to look at programming courses in CreaTe, almost no studies have been conducted for this specific study. However, regular computer science studies can still be examined to gather valuable knowledge. For this purpose, research has been done on the structure and ways of teaching within existing computer science curricula, as well as what makes it difficult for novices to learn how to program. Additionally, more in-depth research has been done on methods that are currently being used to teach novice students about software design within these computer science studies. The methods that were found have been further investigated to find out which difficulties they describe, which solutions to these difficulties are mentioned, and what the authors claim the benefits of using the method are. Lastly, the relevance of these methods to CreaTe is described. Since this project aims to provide students with a fun, alternative way of teaching, this chapter will include a discussion of alternative ways of teaching: informal learning.

2.1 Current state of Computer Science curricula

Programming is seen as a very useful skill, and the demand for programmers has increased rapidly over the years. However, it is hard for novices to learn how to program, and even harder to do this efficiently [28]. According to Winslow [29], it generally takes about 10 years of experience before a novice programmer turns into an expert programmer. However, the aim for graduate outcomes of CreaTe is not to produce expert programmers, and the curriculum is not designed to do so. The aim of this thesis, therefore, lies with novice programmers in a degree that does not aim to teach advanced software development.

2.1.1 Novice programmers

To adjust the CreaTe curriculum, it is important to understand how novices approach programming, and where difficulties lie. As Winslow [29] states, novices often approach programming “line by line”, instead of taking more meaningful structures into account, as well as spend little time on planning and testing code. Their knowledge is often limited to surface level, and they lack detailed mental models. Perkins and Martin [36] mention that, when knowledge is present, a student can still fail to use it (correctly). This kind of knowledge is categorized as “fragile”. Fragile knowledge can take different forms, such as that the student has forgotten it, learned it but does not use it, or misuses it. Soloway and Spohrer [30] and Kurland et al. [35] add that novices in general only comprehend very specific parts of programs. Corritore and Wiedenbeck [34] note that novices understand control flow better than data flow, and can understand beginner operations, such as assignment to a variable, pretty well. Spohrer et al. [37] add that they found most bugs associated with loops and conditionals, in contrast with things like input and output, planning, and syntax.

Du Boulay [33] has created an overview of five domains that a programmer should master, which are non-separable. These five domains are: 1) *general orientation*, where the use of programs is explored. 2) *the notational machine*, which shows a model of the computer related to executing programs. 3) *notation*, which includes syntax and semantics. 4) *structures*, where schemas, plans, and strategies for tackling programming problems are discussed. 5) *pragmatics*, skills such as planning, developing, testing, and debugging. Du Boulay uses this overview to describe the problem of novices often struggling with their first encounters with programming since they try to deal with these different domains all at once, instead of one at a time. Du Boulay also mentions that it is important for students to understand that, while a program should make sense to its creator, the system also needs to understand what it needs to do. Soloway and Spohrer [30] also observed that students are likely to assume that a computer will interpret code in the same way that they understand it themselves, which can be misleading.

However, while the above-mentioned difficulties have been examined extensively in research, it should be kept in mind that there are differences between novices themselves as well. Perkins et al. [38] describe two types of novices: “stoppers” and “movers”. Stoppers tend to simply halt their activities as soon as they are confronted with a problem they do not know how to solve, while movers keep trying, modifying, and experimenting with their code. They also mention extreme movers, or “tinkerers”, who make changes more or less at random when they cannot track the problem in their program. As with stoppers, tinkerers have little chance of making effective progress.

In Section 2.2.1, difficulties of novices are explored further in the context of software design topics within computer science curricula.

2.1.2 Teaching Computer Science

The main way novices learn how to program is via formal instruction, often given during computer science courses. This leads to the obvious conclusion that it is very important to have high teaching standards, as these clearly influence what knowledge students will obtain and how well they will keep this knowledge [39]. Linn and Dalbey [39] propose a “chain of cognitive accomplishments”. These accomplishments should arise from instructions given by computer science teachers. *Features of language* is the first link, followed by *design skills* and completed by *problem-solving skills*. These three links are very important for students to make progress in an introductory programming course, and Robins et al. [28] believe this chain could form what is meant by deep learning when it comes to introductory programming. Regrettably, Linn and Dalbey [39] note that few students actually make a lot of progress in programming during introductory courses.

Linn and Dalbey [39] also mention that laboratory-based programming tasks are an important part of computer science courses. Feedback delivered by compilers and other tools is immediate and consistent, as well as that successfully creating a working program can give confidence to students. Additionally, with laboratory exercises, students can learn at their own pace. Especially for these kinds of exercises, peer learning plays an important role [41, 42, 43], as well as that they provide a beneficial opportunity to students to learn by doing [44, 45, 46]. Additionally, Kurland et al. [40] argue that it can be stimulating for students to use programs

that focus on graphical and animated output. They do note that the emphasis should still be on programming principles in this case.

According to Winslow [29], when it comes to teaching, it is better to keep it simple. Facts, models, and rules can better be introduced in a very simple way, and only expanded upon as soon as the students show signs of clear understanding. He also argues that, while it is important to teach new language features, the combination and use of those features is even more significant. This is especially the case when it comes to program design. Additionally, he states that it is important for computer science courses to distinguish between teaching skills on how to write a program apart from how to read one. Both should be taught, next to basic test and debugging strategies. Next to that, he mentions that different models are important within programming: models of control, data, problems, and design are significant. If a teacher does not explicitly discuss this with their class, students will make up their own models, which are often of dubious quality.

Robins et al. [28] also go into detail about models within programming. They state that having students build a model of a program can help in comprehending it, as well as with the planning, testing, and debugging that come into play when writing the program. However, it should be kept in mind that, while models of a program can be beneficial for the understanding of a student, it is always possible that a student creates a design that incorrectly represents the structure of the program, creating the opportunity for unpredicted errors and bugs. When these bugs or errors are bigger than expected, it could be that the underlying concept needs to be changed drastically. However, this iterative process can also help students in understanding programming better.

2.2 Methods for teaching Software Design to novice students

2.2.1 Difficulties

Several studies have examined existing computer science curricula and the possibility to integrate more software design topics. These studies have made apparent that certain difficulties often surface when examining the existing curricula. Soloway [10], De Raadt et al. [12] and Robins et al. [28] state that there is a focus on syntax and semantics of construct within introductory programming courses, even though these topics did not pose major stumbling blocks for novices. What students often struggle with is the structure of a program. Even when students know their syntax, they do not know where to begin when given a problem statement. Next to that, as Felleisen et al. [15, 17] describe, these students are not challenged to actively explain their thought process behind their programs, leading to students not understanding their own code. In turn, this makes it hard for a student to ask targeted questions and for teachers to get an insight into what knowledge their students have. Soloway [10] argues that one of the causes of this is that students do not have the necessary strategies that are relevant to the problem ready.

Other studies also describe that students rely on prior experience and knowledge when facing a novel problem. Both Castro et al. [9] and Yeh [1] have observed that when students do not have the necessary knowledge ready, they get stuck and find it hard to continue. However, as Soloway and Spohrer [30] state, it should be kept in mind that prior knowledge can also be a source of mistakes if this knowledge is incorrect or applied at the wrong moment. Yeh adds that novice programming students often do not know how to manage the complexity of a programming project. He states that this comes from the practice of grouping programming and software design together as one, while they should be treated as two different types of cognitive activities. They are both essential to software development but serve different purposes. As Yeh states: *“to learn computer programming is to learn the symbolic representations and rules of a programming language; to learn software design is to apply different types of knowledge to various problem-solving situations”* (p. 459). Brooks [31, 32] formulates a similar situation, where he differentiates between the programming domain and the problem domain. What Soloway [10], Castro et al. [9] and Yeh [1], but also Sperber et al. [4], Felleisen et al. [5, 7] and De Raadt et al. [12] state is that topics on the software design part and the accompanying problem-solving skills are often missing within programming curricula. Yeh mentions that some courses do include problem-solving skills but have teaching components that are vague and abstract.

However, making changes to curricula does not come without its problems. As De Raadt et al. [12] state, time needs to be found within the curriculum which can be spent on the new topics. Additionally, when new topics are introduced, their relevance to the industry and the rest of the curriculum needs to be determined. All parties involved with the curriculum need to agree on the addition, which could pose a problem. De Raadt et al. [13] found that Universities frequently force their instructors to spend time on industry-relevant programming languages to attract students. However, these languages are often heavyweight and take quite some time to teach, leaving less time for teaching fundamental problem-solving strategies. Another study done by De Raadt et al. [14] suggests that not only the University influences the time spent on problem-solving strategies: many instructors feel that the programming problems used in their courses are too small for existing problem-solving frameworks, making them avoid teaching problem-solving strategies.

2.2.2 Solutions and their benefits

Every curriculum is different and has specific needs, so it is impossible to determine a solution that is the absolute best. However, there is quite some overlap between the solutions that different articles describe. Sperber et al. [4], Felleisen et al. [5, 7], Yeh [1], Castro et al. [9], Soloway [10], De Raadt et al. [12, 13, 14], Spohrer and Soloway [30] and Winslow [29] all state that novice programming students could benefit from being explicitly taught problem-solving and software planning skills. These skills can help in managing the complexity of programming projects, which in turn will assist students in not getting stuck when writing code. Additionally, a bigger focus on the designing of a program will benefit students in their analytical reading and writing skills. Felleisen et al. [5] argue that these skills cannot be taught by doing drill exercises from a textbook. Instead, they thought of a whole new concept for teaching programming skills. In their project, Program by Design [5], teaching students how to program systematically is

central. The project uses so-called *design recipes* [5], which are explicit instructions on how to come up with a solution to a programming problem. The principle of using these recipes is incorporated in DrRacket, a new programming language Felleisen et al. wrote. To teach these concepts to students, they set up multiple courses with increasing difficulty.

However, not every University has the time to incorporate such a big method into their curricula. Castro et al. [9] identified this time-related problem and started a research that focuses on a lightweight approach. In their research, they examined whether a single lecture on planning and design of software would be enough for students to grasp the concept. First, the students were asked to produce solutions to programming problems, after which they received a lecture on planning and design trade-offs. These trade-offs were meant to show students different possibilities of solving a problem, and why they would choose one over the other. Lastly, students were asked to produce new solutions with different plans, and a preference rank between their solutions. Their results suggest that the single lecture had a positive impact on the ability of students to choose certain solutions over others.

In another study with a different approach, the abilities of students to produce solutions for programming problems were examined. Yeh [1] presented several design scenarios as problem-solving tasks to the participants of the study. Through interviews, the thought processes and proposed actions of the participants were recorded and analyzed. Both novices and experts were included in the research. Yeh found that novices, when asked to verbally express their thought processes, become better at design tasks through a series of tasks over the semester. At first, students would treat features of a program design as black boxes, but at the end of the study, they were able to talk about the functional aspects the feature should have, showing a better understanding of the structure and functionality a program has and/or should have.

2.2.3 Discussion

The importance of better education of software design within programming curricula is clear within literature. While the solutions described differ from one another, many underlying thoughts behind the creation and core aspects of the methods correspond with each other. Each method proposes a kind of systematic approach for programming problems as a solution. While being able to understand your own work is undoubtedly beneficial to the students, teachers can also benefit from this approach. Since students can more clearly describe their problems, lecturers and teaching assistants can distill more easily where common misunderstandings lie.

Another aspect that is in line with this is that it is often mentioned that creating programs is not something you do for yourself: a program should be written to be understandable for others. The skill to be able to do this is quite beneficial for Creative Technologists and is in line with the explicit graduate outcomes. Not only will this help them during their studies, but they will also be more prepared to enter a software career in the future. Whether they will be full-time programmers or serve as a communicator between different disciplines, the ability to read and produce readable code will always be valuable.

Additionally, a bigger focus on software design next to programming will benefit students in their analytical reading and writing skills. Within a diverse study such as CreaTe, these skills are already embedded in the study goals. This would mean that lectures on software

design would not need to have a focus on teaching these specific skills. However, it does indicate that lectures on software design fit within the current curriculum.

For any possible solution, the amount of time within the curriculum it will consume needs to be considered. Since there is limited space within CreaTe for programming, it will be a complicated process to determine where the focus should be and when to teach what. The lightweight approach mentioned by Castro et al. [9] forms an interesting starting point. To get a better overview of what the curriculum needs, more research is needed on what topics are currently included in the curriculum, how much time should be spent on software design, and where within the curriculum these topics could best be taught. This can best be done by examining the curriculum closely and interviewing involved lecturers, students, and alumni.

2.3 Informal learning

2.3.1 (College) Hackathon

According to Warner et al. [16], a hackathon is “*an event where people gather in one location to create prototype software projects within a short period of time, usually from one day to one week.*” (p. 254). Their research focuses on hackathons in a certain location specifically: college. In college hackathons, students form teams to create software projects (“hacks”). These hackathons provide informal learning environments where students can freely try out new things and practice their coding together with peers. Often, these events are sponsored by companies that pay for transportation, food, and prizes in return for exposure and the opportunity to recruit students.

In their research, Warner et al. [16] found that students go to hackathons for both technical and social reasons. The social aspects assist the technical ones: most participants indicate that they learned the most from interacting with their peers. This is also described by Bandura [19] within the social learning theory, which states that learning occurs when people observe and imitate others around them. Another important social aspect mentioned is that less experienced attendees mentioned they felt more comfortable asking questions to their peers instead of teaching staff. This is in line with the observation that students often do not aspire to join the community of practice of college professors. Instead, older students are more likely to be seen as role models, since they have obtained internships at the companies that are present at the hackathon. Additionally, incidental learning takes place, where students learn as a by-product of doing, as well as opportunistic learning: students taking advantage of opportunities provided by the hackathon to self-direct their learning.

This self-directed learning is in contrast to the classical classroom experience, where given subjects are taught in a wide spread-out timeframe. Students mentioned that this specific time dimension of hackathons also posed a clear difference with their regular college classes. Since participants of hackathons are forced to complete their project within a certain number of hours, new knowledge needs to be picked up and applied immediately instead of over multiple weeks. However, a downside mentioned was that this leaves no time for reflection and expert

feedback. Next to that, no grades are given to the projects, making it easier for students to take on something riskier than they would in a regular class.

Lave and Wenger [17] provide a theoretical background for the benefits of hackathons. They describe situated learning, which posits that people are often motivated to learn by doing practical work, especially if this brings them closer to a community of people who are involved with their desired practice. Sakhumuzi et al. [20] mention the same phenomenon within the Social Cognitive Theory. Additionally, Sakhumuzi et al. mention collaborative learning (CL) as one of the core theoretical frameworks behind hackathons. Within a CL environment, students get the opportunity to discuss with peers, present and defend their ideas, are actively engaged in the learning process [21], and can produce higher quality results more efficiently in comparison to working alone [22, 23]. Hackathons provide exactly such an environment. However, it should be kept in mind that Lave and Wenger [17], as well as Guzdial [18] state that situated learning requires the activity to be authentic. This means that hackathons should preferably have a real-life focus, including tools that professionals use.

However, there are also downfalls to hackathons. These events are not mandatory and not every student showed as much interest in going to such an event. Reasons given by participants of the study were: discomfort (bad sleeping accommodations, time pressure), novice fears, no team or idea, no substance (a focus on flashy demos instead of something more substantive), overly competitive ambiance, no time, and hacker culture (discouraging work ethics). Sakhumuzi et al. [20] add as a downfall that participants' intellectual property is not being protected: people do not always work on projects that match their current skill level. Additionally, some people tend to think that organizations use hackathons to get free ideas from developers.

2.3.2 Makeathon

An event quite similar to the hackathon is the makeathon. Many aspects are the same, including groups coming together during an event that lasts for a certain amount of time. However, within makeathons, there is a bigger emphasis on the use of (novel) technologies and the realization of an idea, rather than just coming up with solutions to a problem. Participants make use of maker spaces and mentors to design and build their ideas [24]. Many different makeathons have been set up over the years, each with its own focus and outcomes (see [24] and [25] for examples).

3. Methods and techniques

This chapter describes the methods and techniques that were used for this thesis.

3.1 Iterative process

The project uses an iterative design methodology inspired by Wieringa [49], a University of Twente professor. Wieringa talks about the engineering cycle and the design cycle. The design cycle starts with determining a problem statement, followed by a design and validation of this design. In this cycle, problem-oriented research and solution-oriented research come into play. Problem-oriented research focuses on the real world (learning about artifacts and how stakeholders use them), while solution-oriented research focuses on designing an artifact and validating it. The engineering cycle focuses on the implementation of the design: the design is introduced to the intended problem context in the real world or a test environment. These cycles follow each other: first, a design is determined and evaluated during the design cycle, and afterward, this design is implemented and validated. Wieringa also describes the possibility to nest design cycles if there are multiple subproblems to solve or a sequence of design cycles if the global design needs to be improved. These cycles serve as an inspiration for the iterative design methodology that is used for this thesis.

The iterative design methodology starts with **problem identification** and analysis. A distinction can be made between the analysis of CreaTe and state-of-the-art research in the software design domain. First, the global state-of-the-art when it comes to software design within (beginner) programming courses is analyzed utilizing a literature review. This helps with realizing the extent of the problem and can give insights into possible solutions. Once the broader problem has been analyzed, the current state of CreaTe can be analyzed in detail and a first design of the end product can be made. During this analysis, the **design** goes through multiple iterations as multiple stakeholders are **questioned** and requirements are elicited.

With a first design of the end product, **evaluations** can take place. Stakeholders are also involved in this process. The design goes through another set of iterations influenced by different kinds of evaluations until a satisfactory design has been reached.

This iterative process has been captured in figure 1. The identification and analysis of the problem by conducting a literature review fall under the Identify step. Afterward, the Inquire and Design steps are alternated by questioning stakeholders and updating the design based on the elicited requirements. Once a design of the end product is ready to be evaluated, the Design and Evaluate steps are alternated as the design is improved based on the evaluations that take place.

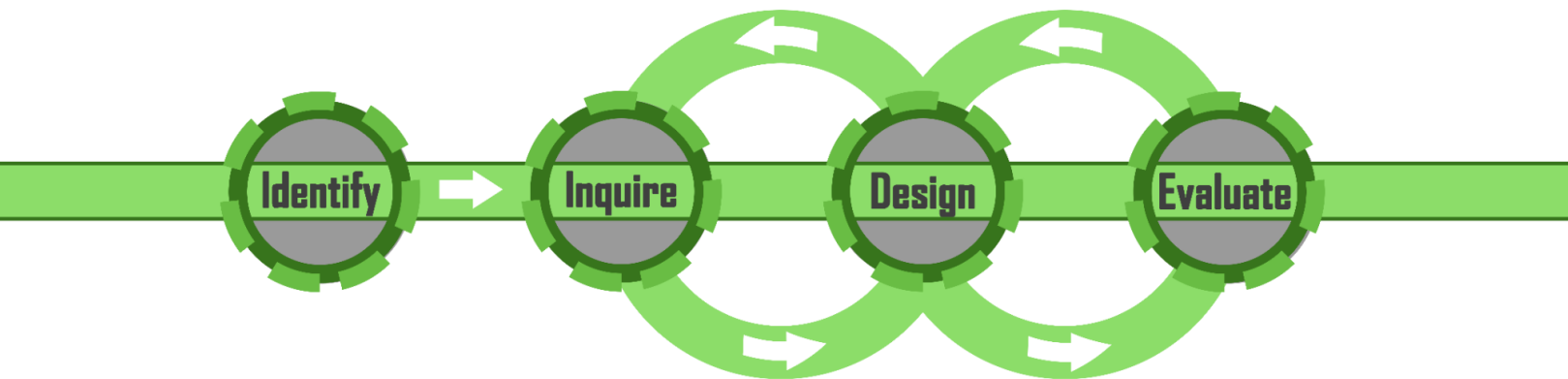


Figure 1: Iterative design process used for this project

3.2 Outline thesis

This thesis describes the different iterations that the final design went through. In the following chapter, Chapter 4, the identification step of the iterative design process is described: stakeholders are identified and the conducted interviews with the teaching staff and alumni are summarized. Additionally, the first iteration cycle of the Inquire and Design step is described. Chapter 5 and 6 both describe such an iteration cycle as well, where Chapter 5 describes how student input influences the design and chapter 6 summarizes a brainstorm with teaching staff. Chapter 7 shows the first iteration cycle of the Design and Evaluate step with results from the evaluation sessions with students, whereas chapter 8 describes evaluation sessions that were held with TAs. Chapter 9 includes the last iteration and the final prototype design. Lastly, concluding remarks are given in Chapter 10 and recommendations for future work in Chapter 11.

It is important to note that the ongoing COVID-19 pandemic influences this thesis and the final product. This means that all activities involving external parties, such as brainstorming sessions, eliciting requirements, and evaluations, need to be hosted online.

4. Iteration 1: Design space exploration

This chapter describes the steps that were taken to reach the first iteration of the project. This iteration contains the exploration of the design space, meaning that design possibilities regarding the project are explored. The exploration includes determining stakeholders, interviews with teaching staff and alumni, and a brainstorm for possible activities. Initial requirements are determined.

4.1 Stakeholders

To determine what the project will entail, stakeholders need to be determined. Stakeholders have a significant influence on what the project will look like in the end since the project is being designed for them. The following table describes which stakeholders are involved with the project and what their role is.

Stakeholder	Role
Hannah Ottenschot	Developer
Ansgar Fehnker	Supervisor
Creative Technology staff	Users
Creative Technology students	Users
CreaTe Programme staff	Decision maker
University of Twente	Decision maker
Developers in the IT work field	Beneficiary/Users
Teachers in the IT work field	Beneficiary/Users
Students from IT studies	Beneficiary/Users
Future employers	Beneficiary
Society at large	Beneficiary

Table 1: Stakeholders of the project and their roles

The stakeholders have been ranked according to how much they are involved with the project. Logically, my supervisor and I are on top of the list since we directly influence the outcome of the project. Followed are the CreaTe staff and students, the obvious users of the project. Next come the decision-makers, as the project will need to be approved by them in the end. They will have a final say in whether or not the project can be integrated with the CreaTe curriculum.

However, while the project aims to be included within CreaTe, this is of course not the only possible outcome. This is why certain Beneficiaries/Users have been included in the list. Developers, teachers, and students who work or study within an IT field could also benefit from this project since it aims to serve an educational purpose in the software engineering discipline. This could either be by actually using the outcome of the project, perhaps in other curricula, as a refresher training, or for further research on the topic. The list ends with two stakeholders who are just beneficiaries, namely future employees and society at large. Future employers will benefit from (CreaTe) students receiving better programming education since these students are their future employees. Society at large can benefit from well-educated IT professionals due to the ever-increasing demand for workers in the IT field [6].

To get an idea of what the project should look like, the interest and values of the stakeholders need to be determined. For the CreaTe technology staff and students, these interests and values are determined by conducting interviews and a survey. Since the Programme staff is aware of University guidelines, the University will not be interviewed separately. What developers, teachers, and students in the IT work field would like to see in the project has largely been described in Chapter 2: Background research and will be taken into account when designing the project. Since it is nearly impossible to determine exactly what future employees and society at large would want to see in the project, the project will be developed without determining the exact interests and values of these parties.

4.2 Interviews

For the first iteration, interviews with CreaTe teaching staff and alumni were held. The outcomes of these interviews have been summarised in a Powerpoint presentation. The respective slides from this presentation are used as clear summarizing overviews of what was said during the interviews.

4.2.1 Interviews with teaching staff

CreaTe teaching staff have a clear overview of what is included in their courses and modules. Their knowledge is very valuable for the project and their opinions will directly influence how the project will take form. In total, four people from the CreaTe teaching staff were interviewed. The questions focused on determining what the opinion of the teaching staff is about the current state of software design within CreaTe, and whether they think CreaTe could benefit from a bigger focus on software design, either within their own module or other modules. Summaries of each interview can be found in Appendix 1.

The results of the interviews have been structured in three categories: what teaching staff believes software design can help with, which obstacles they foresee, and general remarks that came up during the interviews. See figure 2.

Teaching staff opinions

- Software Design helps with:
 - creating a mental model of the project, which will help with increasing code quality
 - Building abstractions, which combat complexity
 - Making code understandable for others
- Obstacles:
 - It's dry material which can be hard to swallow for students, making it harder to sell
 - The importance of Software Design needs to emerge from students themselves to have the biggest impact
 - There is limited time within CreaTe to spend on Software (Design)
- General:
 - There is less Software Design in CreaTe than for example TCS, but there is also a smaller need due to the nature of the study
 - Especially in y2, students should know more about design than is currently the case
 - Important to take different student background into account
 - Software Design should be integrated with programming
 - Next to Software design, good coding habits should be explicitly taught

Figure 2: Positive sides and obstacles of software design and general remarks mentioned during teaching staff interviews

The positive sides of software design mentioned were that it helps with **creating a mental model** of a project: students can more easily imagine in their head what a software project can and will look like. This can help with creating code more efficiently, as well as with understanding the needed programming components of a project. Additionally, software design lends itself to **build abstractions**, which can help in combating complexity. Lastly, a design is most likely easier to understand and explain than regulate code to someone outside of the project, making it **easier to understand for others**.

However, the teaching staff also mentioned that software design includes **dry material** which can be hard to swallow for some students, as well as that it becomes more difficult to convey the importance of the topic. Contrarily, one of the teaching staff mentioned that this is not a problem, as the realization that software design is important should **emerge from students themselves**. Lastly, a big point was mentioned several times: there is simply **limited time** within the study that can be spent on the topic of software design. This is an important point to keep in mind when designing the project.

A general remark that was given was that there is **no need to include a lot of (professional) software design** within CreaTe due to the nature of the study. However, it was also noted that, especially in the second year, **students do not know enough about software design**. It is important to **take students' backgrounds into account when introducing new topics**, however. Additionally, it is preferred to **integrate software design topics with programming concepts**, as this plays to the strengths of software

design. Finally, as a separate comment, it was mentioned that it would be beneficial for CreaTe to explicitly teach good coding habits. This assists students with their own coding abilities, making it easier for them to stay engaged with programming and produce code of good quality.

4.2.2 Interviews with alumni

Since the focus of this thesis is on software design, only alumni who currently were working (distantly) with software design were asked to be interviewed. These alumni have already completed the CreaTe program and successfully found a job related to software design. This means that these alumni can comment on the influence of CreaTe on their knowledge and ability to find a job. This gives insight into the current state of software design within CreaTe and if certain important aspects are missing. In total, four alumni were found who fell into this category and were willing to conduct an interview. The questions for alumni were focused on determining where within their work software design was present, what parts of CreaTe were beneficial to them in this regard, and which aspects were completely new to them. Additionally, the alumni were asked whether they thought the aspects that are currently not within CreaTe should be added in the future. Summaries of each interview can be found in Appendix 1.

The results of the alumni interviews are divided into two lists: all the things alumni would like to keep within the study and what should be added. See figures 3 and 4 respectively.



Figure 3: A summarized list of things alumni mentioned they thought should be kept within CreaTe

The ability to **gain knowledge yourself** was mentioned many times by all the alumni interviewees as an important aspect of CreaTe. Next to that, the skills that come with **working in a team** and **having a specific role** within this team, as well as being able to **translate between different types of work** (programmers, electrical engineers, designers, etc.) were mentioned often. The way Creative technologists can **break down problems in smaller steps** and **think outside of the box** was also brought up. Lastly, the ability to **visualize your ideas**, either digitally, physically, or verbally was noted as important.

Alumni opinions

New

- A bigger focus on real-life practices of working in a company
 - Add **scrum/agile!**
 - **DevOps**, Continuous integration
 - Introduction to **IDEs**
 - Working together on code: **GitHub** practices
 - The steps after writing working code: **testing and release**
 - Making sure code is **understandable and workable** for others
- Technical aspects of a **system** (split between front- and backend for example)
- Bigger focus on **data protection, privacy, and the ethics** behind this (DPIAs)
- UML diagrams

Figure 4: A summarized list of things alumni mentioned they thought should be added to CreaTe

When asked what the alumni interviewees thought could be added to CreaTe, all mentioned that the study currently misses a **focus on real-life practices**. This mainly concerned the **Scrum or Agile** methodology, as well as **DevOps** and **Continuous integration**. Next to that, tools such as **IDEs** and **GitHub** and the process after a piece of code has been written, so the **testing and release** of software were mentioned as important. A general point that was noted is that it is always good practice to teach students that it is crucial to teach students that **code should be understandable and workable for others**. This was also mentioned by the teaching staff and should definitely be included in the project.

One alumni mentioned the **technical aspects of a system**, so the difference between front- and backend, could be added. Additionally, it was mentioned by an interviewee that CreaTe could benefit from a bigger focus on **data protection, privacy, and the ethics** behind this (DPIAs). Lastly, an alumnus mentioned that **UML diagrams** should be a part of CreaTe.

4.2.3 Elicited requirements

Based on the summaries that were made of the alumni and teaching staff interviews, an initial list of requirements was made:

The project should:

- Address **novice software design concepts**
- Focus on **real-life practices**
- Make sure students produce code that is **understandable** for others
- Be **integrated with programming**
- Include topics that **fit** within the current CreaTe program
- Not take up too much **time** from the current curriculum
- Be fit for students from **different backgrounds**

The focus on real-life practices is also substantiated by Lave and Weger [17] and Guzdial [18], who state that learning environments need to be authentic to be effective. This means that a real-life focus should be implemented, including professional tools.

4.2.4 Project outline

With the initial requirements in mind, possible outlines of the project were determined. It was concluded that there are two different shapes the project could take on: a one-time activity or activities spread out over the curriculum. For both options, pros and cons were determined, which can be seen in figure 5.

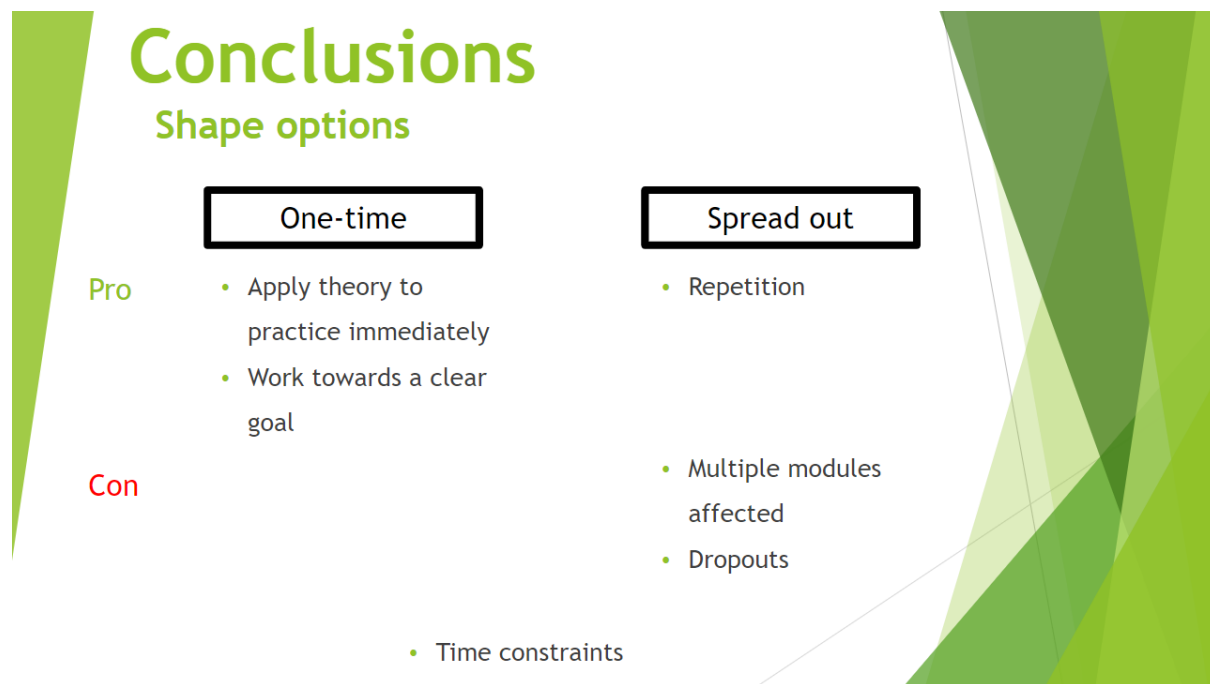


Figure 5: Positive sides and obstacles of software design and general remarks teaching staff mentioned

Shaping the project as a one-time event would make sure that students immediately get the chance to apply the theory they learn to practice. Additionally, you can work towards a clear

goal. However, a one-time event does not have the benefit of repetition. Multiple activities spread out over the curriculum would make sure that students get in contact with the material multiple times, hopefully making sure that they remember the topics. Nevertheless, multiple activities would affect multiple modules, taking time out of several courses. Next to that, if group work is involved, these groups would be affected by students dropping out midway through. For both options, time constraints will always be a challenge.

Additionally, there is the possibility to do a combination of one-time activity and spread-out activities. For example, one-time activities could be given in several modules or a larger one-time activity could be followed by smaller activities in other modules. However, this option is even more affected by time constraints and would need approval from many parties.

4.2.5 Project content

Different possible activities the project could include were determined. These activities and their pros and cons are listed below.

1. Guest lecture

Ideas

#1: Guest lecture

- ▶ Pros
 - ▶ Easy to fit within the curriculum
 - ▶ Can be made mandatory
- ▶ Cons
 - ▶ No practical aspect → small chance students will remember a lot
 - ▶ A suited lecturer will have to be found
 - ▶ Mandatory does not mean students will attend, since you do not get a grade

Figure 6: Activity idea #1: guest lecture, with pros and cons

Since one of the goals is to address novice software design concepts, a guest lecture or multiple guest lectures could be given on the topic. While it is beneficial that this option can be more easily fitted into the curriculum since it would not be incorporated into an existing course, this does mean that no integration with programming courses can take place. Next to that, there is no practical aspect to a guest lecture, which, in general, does not help with the degree to which students will remember the content. A guest lecture can be made mandatory, which could help with conveying the importance of the topic, but making it mandatory does not mean students will actually attend. Next to that, students will not be graded, which can often lead to disinterest.

It was concluded that this single activity would not be suitable to include all requirements. However, it could be one of several activities that are included in the project.

2a. Add software design to existing programming courses

Ideas

#2A: Add Software Design to existing Programming courses

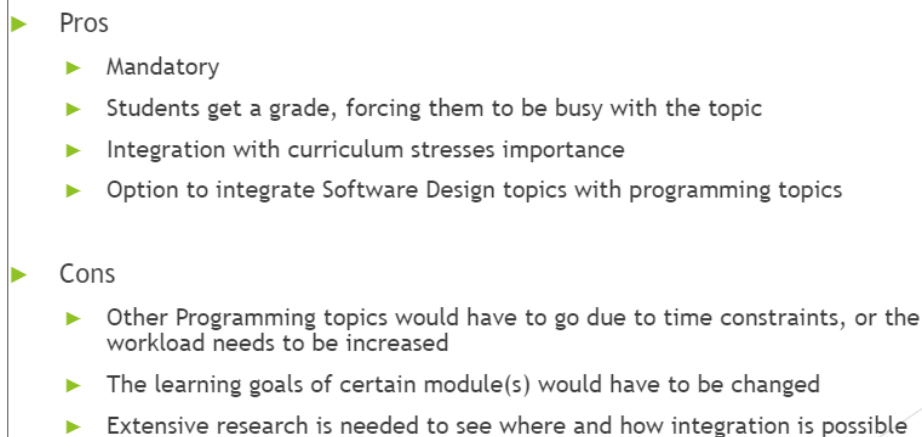
- 
- ▶ Pros
 - ▶ Mandatory
 - ▶ Students get a grade, forcing them to be busy with the topic
 - ▶ Integration with curriculum stresses importance
 - ▶ Option to integrate Software Design topics with programming topics
 - ▶ Cons
 - ▶ Other Programming topics would have to go due to time constraints, or the workload needs to be increased
 - ▶ The learning goals of certain module(s) would have to be changed
 - ▶ Extensive research is needed to see where and how integration is possible

Figure 7: Activity idea #2a: Add software design to existing programming courses, with pros and cons

Instead of hosting a new activity within the curriculum, existing courses can also be altered. As was mentioned during the staff interviews, the option of integrating software design with programming courses was explored. These courses are already mandatory and graded, which forces students to focus on the topics. Additionally, if software design becomes part of the existing curriculum, the importance of software design can be stressed. However, time constraints are still an issue. Additionally, the learning goals of the affected modules would have to be changed to include software design.

2b. Add software design to existing design courses

Ideas

#2B:

Figure 8: Activity idea #2b: Add software design to existing design courses, with pros and cons

Similar to activity idea 2a, software design could also be added to design courses. There are also aspects to software design (interaction, use cases) that could fit existing design courses. However, the integration with programming would be a lot harder, which is disadvantageous. For now, integration with programming courses is preferred over integration with design courses.

3a. One-time workshop(s)

Ideas

“hackathon”

#3A: Workshop(s): one-time

<ul style="list-style-type: none">▶ Pros<ul style="list-style-type: none">▶ A one-time workshop might easier to integrate into the curriculum▶ Working towards one goal like with the M5 hackathon seemed to be loved by many students▶ Theory can be immediately applied in practice▶ Cons<ul style="list-style-type: none">▶ A one-time workshop might not stress the importance of design▶ A hackathon would still need quite some time from the curriculum	<ul style="list-style-type: none">▶ Ideas<ul style="list-style-type: none">▶ Christmas tree idea (let students try to turn on christmas lights)▶ Amsterdam concept (Derive a design from a sentence - every verb is a function, every noun a variable, etc.)▶ M4 preparatory design hackathon for their end project<ul style="list-style-type: none">▶ Could already be started in M3
--	---

Figure 9: Activity idea #3a: One-time workshop(s), with pros and cons

Different possible one-time workshops were determined. In general, similar to the guest lecture, these one-time workshops would be easier to integrate into the curriculum since they are separate units. Additionally, a workshop forces students to work towards one goal as well as that theory can be immediately applied to practice. Last year, the 5th module of CreaTe, Smart Technology, hosted a ‘hackathon’, where students had to create a project using scrap electronics. This hackathon was loved by many, which indicates that the format is approved by students. However, one-time workshops might not stress the importance of software design as much as integration within existing courses, as well as that there still are time constraints to take into account. The different ideas are listed below.

- An idea based on the YouTube video by Matt Parker¹: placing a Christmas tree in the SmartXP with addressable LEDs. Students are given the objective to change the colors of these LEDs to their liking within a day.
 - While this workshop has a high fun factor, the possibilities of adding software design to this workshop might be hard, since the main goal is to turn the LEDs on.
- A concept used by the UVA is to derive the design for a programming project by extracting it from verbs, nouns, and adjectives in sentences. This sentence could be either given by stakeholders or written down by the students themselves.

¹ <https://www.youtube.com/watch?v=v7eHTNm1YtU>

- The concept is solid in theory, but it would need to be combined with other activities to create a full-fledged workshop.
- Lastly, the idea came up to have students prepare for their Processing project in the fourth module by introducing software design concepts. This could be either at the start of the fourth module or already in the third module.

3b. Spread out workshop(s)

Ideas

#3B: Workshop(s): spread out

<p>► Pros</p> <ul style="list-style-type: none"> ► Repetition might help in students' ability to remember the material ► Repetition might stress the importance of design <p>► Cons</p> <ul style="list-style-type: none"> ► Integration of Software Design with programming might be harder ► Spreading out the material might make it harder to estimate students' abilities at certain points ► Students might drop out halfway through the module, making it harder to have students work in groups ► More modules are affected by change 	<p>► Ideas</p> <ul style="list-style-type: none"> ► Have students make designs which they revisit every module ► Different Software Design related workshops in different modules ► Revisit your M1 creature and restructure the code and design
---	---

Figure 10: Activity idea #3b: Spread out workshop(s), with pros and cons

The different activities that involve software design could also be spread out over the curriculum. The repetition of the topics could help with students' ability to remember the material, as well as that the repeated exposure to the material could stress the importance of the topic. However, integration of software design with programming courses would be harder to set up, as well as that multiple modules would have to change their contents. Next to that, it is harder to estimate the abilities of students at multiple points within the curriculum. If the workshop would include project groups, this could also pose a problem, since students might drop out at different points in the study year.

The ideas that came up in the brainstorm are as follows:

- Introduce students to the topic of software design at the beginning of their CreaTe career, and have them make a design that they revisited multiple times during the curriculum.
 - The repetitive aspect would make sure the importance is shown. However, the assignment would need to be of a reasonable size to make sure students really stay busy with the topic and get something out of it.
 - This idea could also be applied to the creature that students have to make in the first module: having them restructure the code and re-evaluate the design they made every module or during certain modules.
- In line with what was said for the guest lecture idea, different workshops could be given in different modules. However, this comes with the same disadvantage as mentioned in the previous idea.

3c. One-time or spread out workshop(s)

Ideas one-time or spread out #3C: Workshop(s):

<p>► Ideas</p> <ul style="list-style-type: none"> ► Students need to create a design for other peers, who will work out their concept <ul style="list-style-type: none"> ► Or: 2nd years will judge 1st years designs ► Online coding challenges <ul style="list-style-type: none"> ► Capture the flag ► Coderbyte.com 	<p>► Pros</p> <ul style="list-style-type: none"> ► The importance of writing understandable code is being stressed <p>► Cons</p> <ul style="list-style-type: none"> ► If the other group does not provide something workable or nothing at all, the group cannot continue ► Transparent grading is needed on whether you are being graded independently of the other group or not
	<p>► Pros</p> <ul style="list-style-type: none"> ► Gamification could motivate students ► Teachers & TAs will have an idea about student progress and who are lagging behind <p>► Cons</p> <ul style="list-style-type: none"> ► Gamification could also demotivate students when they are lower in the ranks ► Quite some administration is needed for such online challenges

Figure 11: Activity idea #3c: One-time or spread out workshop(s), with pros and cons

Some ideas came up that could either take place one-time or could be spread out over different modules:

- Have students create a design, which will be given to other peers who will further work out their concept. Or, have second-year students judge the work of first-year students.

- This concept would make sure that the importance of writing understandable code is being stressed. However, students will be dependent on each other. The concept would have to be made to deal with this. Additionally, a clear and transparent grading scheme is needed.
- Provide students with online coding challenges, such as a capture the flag activity or similar to coderbyte.com. These challenges could be adjusted according to difficulty level, and teachers would have insight into the advancements of the students.
 - A gamification aspect could motivate students to stay on topic. However, this could be dependent on the performance of the student. If a student does not perform well within a ranking, this could have a demotivating effect instead. Next to that, quite some administration is needed to create and maintain these online challenges.

Ideas one-time or spread out

#3C: Workshop(s):

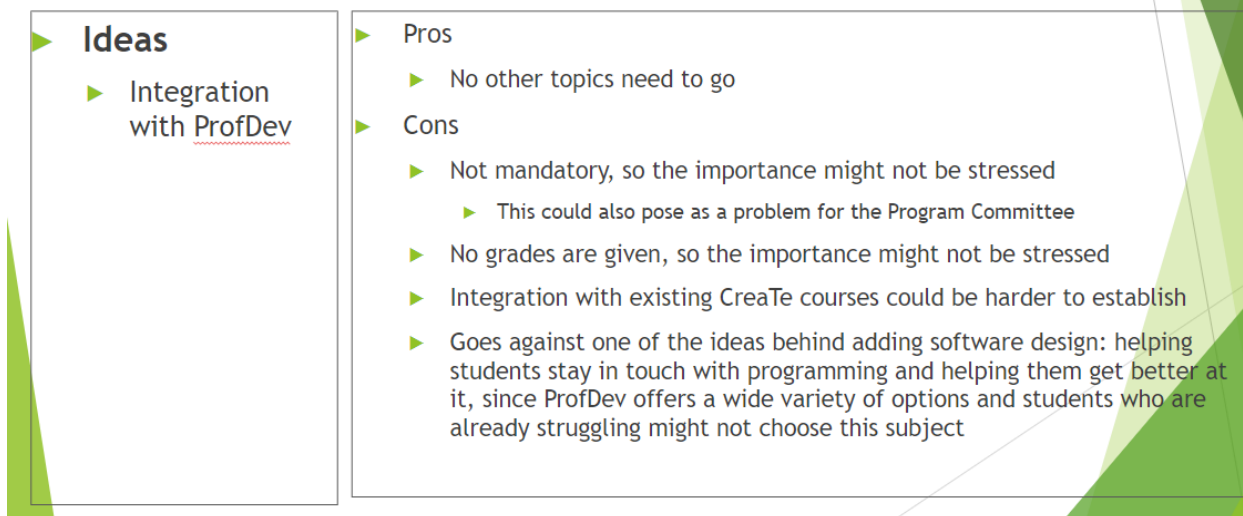


Figure 12: Activity idea #3c: One-time or spread out workshop(s), with pros and cons

- Integrate software design topics with Professional Development. Since Professional Development is its own study unit, integration with the curriculum is a lot easier.
 - However, Professional Development is not mandatory and no grades are given. It would make the topic of software design optional, which is not desirable. Next to that, no integration with programming courses is possible.

4.2.6 Design

Based on the possible shapes and activities that were determined, it was decided that the project includes one big workshop. According to Castro et al. [9], it is possible to make a positive impact on the ability of students by hosting even a single lecture on software design, so it is expected that a single workshop will still be beneficial to CreaTe students. This workshop spans two days, to make sure that all requirements can be included. For the content, two activities are incorporated: the UVA concept where students need to elicit a design based on sentences and the idea of having students judging the designs of other students. Additionally, the workshop focuses on students being able to write quality code that is understandable for others.

The first draft was originally made in Dutch and can be found in Appendix 2. The translated version can be found below.

Day 1

(9:00 - 10:30) The first day begins with a short introduction to software design and why it is important. Next to that, the first assignment will be explained and some example outcomes will be given.

(10:30 - 12:30) Afterward, students will split up into groups of 2 to create a design. They will do this by breaking down sentences. Roughly, nouns indicate classes, verbs indicate functions that need to be included and adjectives become variables. The design students create will be checked and graded by TAs.

(13:45 - 15:00) As soon as the design has been approved, students need to write code based on their own design. First, a short presentation will be given on common mistakes programmers make. Preferably this will be done during a live coding session, where the teacher will explain their thought process behind going from a design to code, as well as the actual writing of it. Common mistakes will be clearly shown, as well as ways to avoid them. After this lecture, the second and third assignments will be explained, with the notion that the code students are going to write should be understandable for others, and a short introduction to GitHub will be given.

(15:00 - 17:30) Students can work on their code. GitHub is recommended for this. If students come across any problems, they can ask TAs for help. The deadline for handing in the code is at 22:00 the same day.

Day 2

Before 9:00, it should be determined which teams will give their code to each other.

(9:00 - 9:30) Code is exchanged (the students do not get the chance to explain what they made - the code should be self-explanatory).

(9:30 - 12:00) Students start their third assignment, where they will create a design based on the code they received from another group and provide the group with a list of recommendations. This design and the list of recommendations should be approved by a TA.

5. Iteration 2: Student requirements

With the first version of the introductory workshop manual ready, it was time to improve the concept. This chapter describes the influence of the student survey on the manual.

5.1 Survey

A survey was sent out to all CreaTe students and alumni. The survey was meant to determine what was the opinion on the Programming courses within CreaTe, as well as to establish the perceived knowledge of students about software design at this point in their study career. The opinions of students are very important to the project since they can help with understanding what CreaTe has offered its students and how students perceive this. Their comments will directly influence the shape and design of the workshop and the list of requirements can be updated according to what students have to say. The participants were asked to state how far along they are in their CreaTe career, as well as in which academic year they started with CreaTe. This was done to make sure that all answers on specific modules or courses could be related to the academic year they took place in. In total there were 47 responses, of which 9 first years, 9 second years, 20 third years, and 9 alumni. The complete summary of the survey can be found in Appendix 3.

5.1.1 Survey results

Participants could fill in the survey from April 8th, 2021 to May 8th, 2021. From the survey, several interesting points can be noted. The survey is processed per question.

Q1: During which Creative Technology module did you learn most about programming?

Three modules are mentioned often: module 1 (10 participants), module 4 (10 participants), and module 6 (11 participants).

It is interesting to note that 2 second-years and 2 third-years filled in module 1 as the module they learned most from, while it would be logical that these students learned more from later modules. It could be the case that these students filled in this answer since they learned their foundational knowledge in the first module, or because they found it hard to keep up in later modules.

An unexpected find was that two participants answered module 3, while later on in the survey it becomes clear that many people think module 3 lacks programming.

Second-years mentioned module 4 most often (4, 44% of all second-years) Third-years mentioned module 6 most often (7, 35% of all third-years).

Another interesting point is that, from the participants who filled in module 6, only 2 second-years (22.2 %) wrote this as their answer, while this was the most popular answer for third-years. Of these two second-year students, one started in the academic year 2018-2019 and thus most likely is redoing the module. The low number of second-years could be because module 6 was still not finished for quite some time when the survey could be filled in, but could also be because of changes in the module content.

Q2: Do you feel confident in your programming skills? (rate 1 - 5)

On average, students rated this question with a 3.11 out of 5.

A very interesting point is that the average scores show a bell curve: first-years score a 3 on average, second-years score highest with a 3.44, but then the scores lower with a 3.25 for third-years and a 2.78 for alumni. While this could of course just be a coincidence or shows differences in ability or module content for the different years, it could also be a sign of the Dunning-Kruger effect. This effect is a hypothetical cognitive bias that states that people who do not have many skills yet are quick to overestimate their abilities [47]. Since first and second-years do not yet know to what extent their knowledge could go, it could be that they feel as if they already know a lot and feel confident because of this. Additionally, students often compare themselves to their peers. The peers of alumni are often professionals with other backgrounds, so this might be why alumni feel less confident in their programming skills.

It is good to note that all participatory groups score above 2.5, indicating that most students do feel confident to some extent.

Q3: To what degree did the Creative Technology program assist you in becoming confident in your programming skills? (rate 1 - 5)

On average, students gave a score of 3.04 for this question.

It seems that the first and second-years feel quite a bit more supported by CreaTe than the third-years and the alumni: the first-years scored 3.67 on average and the second-years 3.56, while the third-years scored 2.65 and the alumni 2.78. This is most likely because quite some students learn programming during their minor or at work, so outside of CreaTe, as well as that modules 11 and 12 do not include any programming.

Similar to the previous question, the average score is above 2.5, so most students do feel assisted in becoming more confident in their programming skills by CreaTe.

Q4: To what degree does your confidence in your programming skills influence your motivation to write code? (rate 1 - 5)

On average, students rated this question with a score of 4.17

First-year students seem to give the lowest score, while third-year students score the highest. This could be because the first year of CreaTe includes more mandatory programming, meaning that their motivation has less influence on the amount of programming they will do in the end.

An average score above 4 indicates that students do think confidence makes a big difference when it comes to feeling motivated to write code. This means that it is very important to make sure students feel supported by the CreaTe curriculum in their confidence in programming skills.

Most remaining questions are open questions. Only the most given and noteworthy answers are noted.

Q5: Which factors of Creative Technology had the greatest impact on your confidence in your programming skills?

A big portion of the participants (9 participants, 19.15%) noted that they learned a lot from (completing) the bigger end projects.

5 participants (10.64 %) mentioned they think practicing is very important. Completing goals can feel very motivating, and it is important to keep coding and practice your skills.

3 participants mentioned certain courses or activities: Algorithms, AI Theory, Programming Sports Day, and 2 participants mentioned Python specifically.

Some participants mentioned that practical technicalities, such as getting a grade, making mandatory (weekly) assignments, and individual assignments made a difference.

2 participants mentioned they got scared to ask questions during the first module since they quickly fell behind.

Q6: Are there other factors (outside of Creative Technology) that influenced your confidence in your programming skills?

11 participants (23.40 %) mentioned that doing personal programming projects made a big difference.

7 participants (14.89 %) noted they did one or several modules from Technical Computer Science which had a big influence on their programming skills, and 3 participants mentioned they learned a lot from acquaintances who did Computer Science.

4 participants mentioned they worked on game development in their spare time, and an additional 3 participants mentioned participating in game jams. This indicates that there are quite some students within CreaTe who like game development.

4 participants mentioned watching YouTube videos in their spare time.

4 participants mentioned being a TA (teaching assistant) during programming courses.

3 participants mentioned they learned some programming in high school.

Q7: Do you feel as if there are any modules that currently lack education on programming? If so, which one(s)?

20 participants (42.55%) mentioned they think module 3 and module 7 form big gaps regarding programming content and need a bigger focus on programming. This is quite a big percentage, indicating that modules 3 or 7 are a logical option to include more programming. Modules 1, 2, 4, 5, and 8 are also mentioned sometimes as modules that could use more programming, but less often than modules 3 and 7.

4 participants mention they think the amount of programming is good.

Q8: Are there any modules that already have too much programming in it? If so, which one(s)?

2 participants mentioned that module 4 went rapidly from topic to topic, making it quite a hard module considering the amount of programming knowledge students have up to that point.

2 participants mentioned the leap from Processing to Python in module 6 is quite a big one.

2 participants mentioned they like programming, so they would prefer more over less.

Q9: If you could change the topics on programming that are treated within the Creative Technology program, what would you change?

12 participants (25.53%) mentioned they would like to see a bigger focus on language fundamentals (how does Java work, syntax, documentation, best practices) and problem-solving skills. This is in line with the background information that was gathered in chapter 2: Background Research, so that is quite interesting to note. It is at least an indication that the proposed workshop will cater to the wishes of students.

6 participants mentioned they would like to see more Python within the curriculum.

5 participants mentioned they miss information on the design of software systems and 4 participants mentioned they would like to see a bigger focus on software engineering skills (knowing software lifecycles, using agile development using git, how to test your software). This is also a clear indication that the focus of the proposed workshop is in the right direction.

4 participants mentioned that they think CreaTe should focus less on visual programming (Processing) since this often does not translate well to programming jobs. This shows that students do think about the connection between the languages they learn and their job perspectives.

Q10: Do you have any other comments related to this topic you would like to share?

2 participants mentioned that CreaTe programming feels a bit like a crash course. While this is not a problem in itself, it does create an awkward divide between people who want to become better at programming and those who are not interested. The programming courses should have to cater to both. This is something the workshop should also take into account.

2 participants mentioned that it is quite a problem that, if you work together on a programming project and one of the group members is not good at programming, it is very easy to just lift off of the person who is. This is an interesting point to take into account for the workshop.

2 participants specifically mentioned taking inspiration from the Design course in module 2 of Technical Computer science would be a good idea.

Q11: Do you feel as if you know enough about software design, taking your study career into account?

More than half of the participants (27 participants, 57.4%) answered “no”, which is a clear indication that students do not feel confident in their knowledge of software design.

14 participants (28.9%) answered “yes”.

2 participants mentioned they learned a lot about software design during their minor in Computer Science.

Q12: Do you think Creative Technology should include more software design related topics in the program? If so, where within the program should these topics be treated?

10 participants mentioned they think learning how to sketch out a program before starting would be a great addition to the curriculum. This is beneficial for the current course the workshop is taking.

10 participants mentioned they think it would be good to add software design early in the curriculum, so students can get the most out of it in their further careers.

4 participants mentioned that they think software design can help with understanding a system.

3 participants mentioned module 3 could use more software design since you get a use case for the first time.

The answers for this question in combination with the first questions indicate that the choice of module 3 as the moment to host the workshop is in line with the wishes of the students.

Q13: Do you have any other comments related to this topic you would like to share?

1 participant mentioned that the focus should not be on UML diagrams, but on the structure of your code. This is in line with what the teaching staff had to say during their interviews.

1 participant mentioned that students should deliver a design before they start working on their program.

5.1.2 Updated list of requirements

Based on the student survey, several new requirements can be determined. Some of the existing requirements are also endorsed by the participants of the survey, which is good to note.

Next to the original requirements, the introductory workshop should:

- Preferably take place in **module 3**
- Help students become **confident in their programming skills**
- Include a relatively **big project**
- Focus on **language fundamentals** and **problem-solving skills**

An interesting point was raised by participants about whether assignments should be completed in groups or not. If a student works together with someone better at programming, it can happen that all the work will be done by one person while the other learns nothing. Due to this, it was considered to add a requirement that the workshop should only include individual assignments. However, due to the size of CreaTe, this is nearly impossible to host, taking into account that all work of the students needs to be checked by TAs. Based on this, it was decided that the workshop will be hosted for groups of 2 instead. Additionally, hosting the workshop for groups of two could be beneficial for students due to peer learning. Kurland [40], Van Gorp & Grissom [41], and Williams et al. [42] all describe that peer learning plays an important role in how well students will be able to understand and complete assignments. This is also described by Bandura [19] in the social learning theory, which states that learning occurs when people observe and imitate others around them. On the same topic, Sakhumuzi et al. [20] mention collaborative learning (CL). If students get the opportunity to discuss with peers and present and defend their ideas in a CL environment, they are actively engaged in the learning process [21] and can produce higher quality results than when working alone within a shorter amount of time. Warner et al. [16] also found that students feel more comfortable asking questions to their peers instead of teaching staff, which is another reason why having more than one student work on a project is beneficial.

6. Iteration 3: Workshop manual

After the opinions of students were established, a new iteration of the workshop manual could be made. The teaching staff is involved to give feedback on this new iteration and create a final version of the workshop manual.

6.1 Brainstorm with teaching staff

The second iteration of the introductory workshop manual was discussed with two members of the teaching staff. In general, they were quite enthusiastic about the current layout of the workshop. They agreed with the use of Processing since students know this language and found it beneficial that theory can be applied immediately to practice. The use of Processing is also backed up by Kurland et al. [40], who argue that programs that focus on graphical and animated output are stimulating for students. The teaching staff also thought it was a good idea to use GitHub since it will be used again at least in module 6, and can optionally be used in module 4.

There were some suggestions to build on the current version of the workshop:

- It was suggested to add more examples to the manual.
- To make sure students will include interaction clearly in their diagrams, it could be considered to focus slightly more on the interaction part of a program.
- For the second and third assignments, it was suggested to split the project teams into two groups, who will each receive a different version of the second assignment. This makes sure that for the third assignment students who had to write code for the first version of the assignment will have to assess code for the second version of the assignment. By doing this, students do not have to give feedback on the code they have been working on for the past hours.
- For the third assignment, it could be considered to include that students explain their feedback to the receiving group verbally since textual feedback can be interpreted in different ways.
- It was mentioned that students from Technical Computer Science are working on a user-friendly application for making all kinds of UML diagrams. This program could maybe be used for the workshop.
- It was suggested to make a document per assignment with clear learning goals, explanations, and examples.
- To make sure students get enough time to grasp the concept of software design, it was mentioned that more time could be added to Lecture 1.1.

6.2 Final version of the introductory workshop manual

Based on the brainstorm with the teaching staff, the final version of the introductory workshop manual could be made. The biggest change is that the assignments and lecture descriptions were worked out in more depth and were provided with examples. Next to that, the schedule was updated, the learning goals of the workshops were worked out and the draft was given a better layout. The final version of the introductory workshop manual can be found in Appendix 4.

7. Iteration 4: Evaluation workshops

The workshop described in the manual takes place in Module 3 of CreaTe. This means that it is impossible to evaluate the workshop in a real-life setting during the time frame of this graduation project. To overcome this, an evaluation workshop was designed. This evaluation workshop is based on the manual and its purpose is to evaluate the idea of the original workshop with students. Since it was expected that no volunteers would be willing to spend the 32 hours needed for evaluating the complete workshop, it was decided that the evaluation workshop should take up a lot less time. However, this does mean that not every aspect of the original workshop can be evaluated using this evaluation workshop, or at least not as in-depth as was preferred. This chapter describes the evaluation workshop, as well as the reasoning behind why it is justified that certain aspects of the original workshop are not included or not given as much attention as in the evaluation workshop.

7.1 Evaluation workshop

7.1.1 Layout

Because the amount of time for this workshop would make a big difference in the number of participants willing to join an evaluation session, a time limit was determined first. To align with the schedules of most students, it was decided that the workshop should not take up more than 3 hours. This meant that evaluation sessions could be hosted both before the break and after while still staying within work hours.

With this time limit in mind, the lectures and assignments could be decided on for the evaluation workshop. The chosen lectures and assignments were included in the schedule that was made. This schedule can be seen in table 2.

Activity	Amount of time
Intro to software design	10 minutes
Intro to assignment 1	5 minutes
Work on assignment 1	30 minutes
Explanation of original workshop	5 minutes
Intro to assignment 2	5 minutes
Lecture on good coding habits	10 minutes
Work on assignment 2	30 minutes
Discuss feedback for second assignment	10 minutes
Closing & evaluation	15 minutes

Table 2: Schedule for the evaluation workshop

With this division, the evaluation workshop would only take up to two hours. The content of the lectures and assignments are described in subchapter 7.2: Content of evaluation workshop. It can be noted that some lectures and assignments from the original workshop are missing. The reasoning behind why this is justified can be read in subchapter 7.3: Justification of evaluation workshop.

To gather participants for the evaluation workshop, a Google form was set up. In this form, participants were given the choice of either signing up alone or in a group of two. Additionally, five different time slots of two hours were presented as options. Participants who filled in the form could indicate during which of these five time slots they were available. Lastly, participants were asked to leave their contact details, so I could contact them about the time slot they were assigned to and give them additional information. The form was sent out to all CreaTe students via WhatsApp. It was decided that a diverse group of students should comment on the ideas for the software design workshop to get the most useful feedback. Additionally, it would have been quite hard to gather enough first-year students alone to join the evaluation workshop.

7.1.2 Content

The slides for the presentations and the content of the evaluation workshop can be found in Appendix 5.

The **introduction to software design** and **assignment 1** presentations were made based on the manual. The content has been redesigned to fit within the time limits of the evaluation workshop.

After the second lecture, the evaluation workshop deviates from the **original workshop**, since the second assignment of the original workshop has been left out of the evaluation workshop. Because of this, it was decided that a small presentation would be given at this point to explain to the participants what the original workshop looks like. This could help them put the evaluation workshop in perspective and give more concise feedback after the workshop.

The second assignment of the original workshop is not included in the evaluation workshop. It was still decided to include a presentation on **good coding habits**. The participants of the evaluation workshop do not have to write code themselves anymore but do still need to evaluate and analyze existing code during the second assignment of the evaluation workshop.

Lastly, a presentation is given introducing **assignment 2**. This assignment is very similar to the third assignment in the original workshop. However, for the evaluation workshop, participants are given code that I wrote instead of another participating team. This code intentionally contains structural problems and questionable design choices. Participants still need to make a design based on the given code, again including the three different diagrams, and feedback needs to be provided as well. Instead of discussing the feedback with their peers, participants are asked to discuss their feedback and recommendations within the group of participants.

After the participants have finished discussing the second assignment, there is time to give feedback on the evaluation workshop, as well as on the ideas for the original workshop.

7.1.3 Justification

The evaluation workshop is missing some of the aspects that are included in the original workshop. However, the evaluation workshop has been set up in such a way that it is still justified to use this evaluation workshop to evaluate the original workshop as a whole. The changed and missing parts are listed below.

Addition of the ‘Original workshop’ lecture

Since the evaluation workshop is designed to evaluate the original workshop, it is logical to let the participants know what the plans are for the original workshop. Because of this, it was decided to add a small presentation on what the original workshop looks like. This presentation can also be used for the final evaluation round.

Removal of the live coding session

Due to time constraints, the live coding session was removed from the evaluation workshop. It is quite hard to include such a session if the workshop only takes 2 to 3 hours. However, a lecture on good coding habits is still included in the evaluation workshop. This makes sure that participants of the evaluation workshop are still given an idea of what students would be taught during the original workshop and gives them the necessary information needed to complete the second assignment.

Assignment 2 of original workshop removed

The original workshop includes three assignments, while the evaluation workshop only includes two. The original second assignment, where students program based on their own designs, has been removed. The decision to remove this assignment had to do with the set time limit and the attractiveness of joining the evaluation workshop. For a workshop set out for all first-year CreaTe students, it is logical to include programming and this does add quite some value to the workshop. However, it is impossible to cater the evaluation workshop to the programming level of all participants, since they will most likely not exclusively be first-year students. There would be only around 30 to 45 minutes available to spend on writing code, which is not a lot of time. Participants would produce software of varying quality, making it nearly impossible to effectively host the third assignment, as well as evaluate the second assignment. Additionally, not everyone is as good at programming. For someone who is doubting whether they want to join an evaluation workshop, especially if they think they are not as good at programming, having to program can be the deciding factor to not join. Without the programming aspect, the evaluation workshop can still function as an effective evaluation of the original workshop. After all, the intended outcome of the workshop is not to teach students how to program, but rather about software design in general and the ability to communicate about code. Participants still need to create a design based on existing code and give feedback on it, so the decision to remove the second assignment is justified.

Decrease of time spent on all subjects

The biggest change that was made when creating the evaluation workshop, is the amount of time that is spent on all parts of the workshop. But, as explained before, it was simply not doable to spend more time on this evaluation workshop, as it would have resulted in no participants and thus no feedback, destroying the whole purpose of the evaluation workshop. All lectures and assignments for the evaluation workshop were designed in such a way that the complexity matches the time available. For example, the introduction to software design lecture only takes 10 minutes but includes only the aspects necessary to get an idea of what software design is and to complete the two assignments. Since the evaluation workshop is not meant to teach participants about software design, but rather to make sure they have enough knowledge to correctly evaluate the ideas surrounding the original workshop, the decrease in time spent on all subjects is legit.

Working alone instead of in groups of two

The evaluation workshop gave the option to participants to either sign up alone or in groups of two. This was done because it was expected that quite some possible participants would get demotivated by the fact that they would have to look for a partner to work together with. Additionally, the original workshop was designed to include groups of two students mainly because of the inability to provide enough TAs and feedback to more than 100 students who work individually. Some students indicated in the student survey that they would prefer individual programming assignments over group assignments since it forces you to focus on the subject. This is enough reason to say it is justified to host the evaluation workshop for individuals instead of groups of two.

7.1.4 Results

The Google form in which participants could sign up was distributed among CreaTe students and alumni and was available from May 19th, 2021 until May 22nd, 2021. During this time, 7 participants signed up for the evaluation workshop and a pilot session was held with one additional participant.

7.1.4.1 Pilot session

During the pilot session, practical things such as hosting the workshop on Microsoft Teams and timing were walked through. Additionally, the content of the presentation slides was checked for inconsistencies and unclarities. Some spelling mistakes and numbering issues were resolved. No large issues within the evaluation workshop were found. However, one thing was changed: the explanation of the original workshop was moved to the beginning of the evaluation workshop. This would make sure participants could keep the original workshop in mind during the whole evaluation workshop. The participant also had some suggestions for and comments on the original workshop:

- For the live coding session, it is important to include the classroom. Have them make suggestions on what the teacher should do next and why, after which the teacher will explain whether this is a good or a bad idea.

- Make sure that the lecture on good coding habits is given before the live coding session, so people can see the habits in practice.
- The fact that students have to do the same assignment, creating three different diagrams, twice was discussed. In the original workshop, there are quite some hours between the two assignments involving the creation of these diagrams. But, during the evaluation workshop, this is only about 20 minutes. This can make the assignment feel very repetitive. Additionally, there are only 30 minutes for the second assignment of the evaluation workshop. If participants need to both create these three diagrams again and have to provide feedback on code, there is a chance they would not be able to complete the task. It was decided that it would be tested during the first evaluation session whether it is possible to complete both the diagrams and to give feedback during the given time frame.

The participant made a diagram for the first assignment. It can be seen in Appendix 6.

7.1.4.2 Evaluation workshop results

The 7 participants were divided among time slots, according to their availability. The workshops that were given were the same every time, except for small changes that were implemented based on feedback received during the workshops. The received feedback will be discussed grouped in feedback on the lectures, feedback on the assignments, overall feedback, and comments on the content of the original workshop.

Workshop 1

2 participants were present during this workshop.

• **Lectures**

The introduction to the software design lecture was clear and gave a good quick insight into the basics of software design. Additionally, it was mentioned that the chosen diagrams are suitable for beginners since they are easy to understand and can help form a mental model of software. One of the participants mentioned they liked the use of the diagrams since they help students get used to a certain thought process that is useful when writing code.

The lecture on good coding habits was received well, but the participants agreed that some aspects were missing: coding conventions, getters, and setters and clear examples showing good vs bad. These aspects will be added to the lecture in the original workshop.

• **Assignments**

A small mistake was noticed in one of the lecture slides for the first assignment, which was changed for the next workshop. One participant mentioned they found it hard to complete this assignment in time, especially since they thought for a long time about the data flow within the system and found it hard to include this in a class diagram. This participant did agree that, if they would have had the time to make an interaction diagram, this could have helped. One

participant also mentioned that creating the interaction diagram helped with revising the class diagram since it became easier for new ideas and possible issues to arise. However, it was also mentioned that the use case diagram was not as useful for this evaluation workshop. Nevertheless, it was agreed upon that such a diagram could be useful if the complexity of the assignment is increased, which is the case for the original workshop.

For the second assignment, it was rightly pointed out that it would be nice to have a visual representation of the code. A picture of the running code was added to the code file for future workshops. It also turned out that some comments in the code were left over from previous versions and were incorrect. While it is also good to see if students notice these kinds of mistakes, it was decided to change these comments to correct ones to make sure the participants could focus on other parts of the code. Additionally, no participant was able to produce both feedback and three diagrams, so it was decided that the creation of the diagrams will be removed from the evaluation workshop.

- **Overall**

Overall, the participants liked the evaluation workshop and learned something from it.

- **Original workshop**

The participants were positive that the workshop would have added value to the CreaTe curriculum. They agreed with the additional lectures and assignments. One participant commented that it would be a good idea to add a lot of time to the assignments where students can ask questions to a TA. For the second assignment, students must get the opportunity to create quality code, and for the third assignment, students should get enough time to discuss their feedback, to make sure the given feedback is of quality.

Workshop 2

During this workshop, 3 participants were present.

- **Lectures**

There were no comments on any of the lectures.

- **Assignments**

One participant mentioned that it is quite hard to create three diagrams in just half an hour if you are not experienced with UML. However, it was concluded that this participant thought all diagrams should be made using correctly written UML since the examples show UML as well. It was decided that it should be clearer for the final evaluation workshop and original workshop that UML is not needed to complete the assignments. This could be realised by adding more informal examples, such as hand-written diagrams. Another participant mentioned it is also hard to create three diagrams if you have not programmed in a while. Contrary to the previous evaluation workshop, it was concluded that 30 minutes is not enough time to complete the first assignment. However, it was still decided to keep the amount of

time spent on the first assignment, since it was not possible to increase the amount of time for the evaluation workshop anymore.

- **Overall**

Overall, the participants liked the evaluation workshop. One participant mentioned he did not feel as if there was new information in the workshop compared to what they learned within CreaTe. This was quite surprising since most of the comments from other students suggested otherwise.

- **Original workshop**

One participant mentioned they especially liked the idea of having students evaluate other students' code. This emphasizes the importance of clear code and gives better insights into the abilities of peers. An idea that came up during the evaluation was to have students write the textual prompts for other students as well. This would make the material a little bit less dry. However, it would increase the time teachers and TAs would have to spend on logistics. Next to that, it would be difficult to make sure all textual descriptions are of the same difficulty. This would take quite some time from teachers and TAs as well. Another point that came up during the workshop, was that it should be assessed which parts of this workshop should (only) be included in the workshop, and which parts should be included elsewhere in the curriculum. It was discussed that it would be good to already include some software design in the first module, and repeat certain aspects in later modules.

Workshop 3

2 participants were present during this workshop.

- **Lectures**

Some content-related questions arose during the introduction to software design lecture. The participants mainly needed some additional explanation or affirmation of their understanding. It was agreed that no changes were needed to the presentation since the questions were very similar to the kinds of questions a student would ask during a normal lecture.

For the lecture on good coding habits, it was again mentioned that examples would be beneficial. Additionally, one participant said they would like to see common misconceptions within programming in this lecture. Luckily, it was already the plan to add this to the live coding session and lecture on good coding habits of the original workshop.

- **Assignments**

The participants found it challenging to finish the first assignment within the given time. It was again mentioned that this was mainly due to the fact of not having programmed for a while, but also since the material was new and some time was needed to grasp the content. Additionally, it was mentioned that it would be a good idea to give the students multiple examples for the first assignment. This would make sure that students can see different

possible outcomes of the assignment, as well as that they would not simply copy what is on the single example. One participant mentioned that the diagrams reminded him of Scratch, a visual programming language, and suggested looking into this for the original workshop. However, since Processing is the only programming language some CreaTe students are familiar with, as well as that Processing is used later on in the study, it was decided not to add another programming language to the original workshop.

For the second assignment, one participant noted that certain mistakes were repeated many times and that it might be better to add more different mistakes. However, in the original workshop, students do not receive code from the teaching staff, but from students. Since this evaluation workshop was the last one, no changes were made to the provided code for the second assignment. Next to that, it was mentioned that having the actual code as opposed to receiving a pdf would be preferred. This would make sure students can execute the code themselves and switch between the different class tabs. This will be added to the original workshop.

- **Overall**

The participants mentioned they liked the evaluation workshop and learned something from it.

- **Original workshop**

The participants agreed that the workshop would be a good addition to the CreaTe curriculum. It would provide a structural basis for software design, which can help students with their programming skills. One participant questioned whether module 3 might even be too late since the basis would be best to provide as soon as possible. Since most students who participated in the student survey agreed that module 3 would be a suitable spot for the workshop, the workshop will most likely stay within the third module. However, it could be looked into whether certain parts of the workshop could already be mentioned within earlier modules and/or repeated in later modules.

8. Iteration 5: TA validation

An evaluation meeting was set up with TAs (teaching assistants) and teachers from module four to evaluate the ideas for the software design workshop since their experience as a teacher or TA can give new insights into the possible success of the workshop. Previously, only the students and some teachers were included in the design process, but TAs will play a significant role during the workshop. A presentation was created to guide the evaluation process. The presentation focuses on the aspects of the workshop TAs need to help, as well as evaluating the amount of time that is spent on each assignment. Next to that, the lectures and the amount of time allocated to them are presented. The presentation slides can be found in Appendix 7.

8.1 Evaluation with TAs

In total, there were 6 participants, of which 4 TAs and 2 teachers. Some participants had to leave earlier due to other obligations. The participants were sent the manual for the original workshop to prepare for the evaluation session.

The evaluation session started with a short introduction to what the workshop entails and why it is being designed. Afterward, the necessary content of the manual was explained so the TAs could get an idea of the workshop.

Overall

The participants were asked what they think of the structure of the workshop, as well as whether they think the workshop is worth adding to CreaTe. All participants were positive about the idea and thought that the subjects covered could be beneficial to the curriculum. There were some comments and suggestions.

- Some questions arose about the decision to host the workshop in module 3. After explaining that this choice was made based on the opinions of a significant number of students, everyone agreed the placement is suitable.
- The use of UML was discussed. It was agreed that UML should have a place within CreaTe, but that it should be lightweight. It is an important aspect of teaching about software, but there is not enough time within the curriculum to cover all there is to know about UML, especially if teaching UML would be combined with teaching about correctly structuring code and using good coding habits. Next to that, it was mentioned that UML is not as unified as it might seem at first. If CreaTe would want to cover all of UML, a selection would still have to be made on which kind of UML would be taught.
- It was mentioned that UML diagrams, especially the class diagram, can be quite beneficial for CreaTe since they can help create an overview and provide students with a visual representation. However, some UML diagrams, such as the state machine diagram, are not as useful and do not need to be covered.
- As of now, there are three lectures (2.1, 2.2, and 2.3) scheduled right after each other. It was mentioned that it might be nice to break up this stream of information with something fun and interactive, to help students pay attention during all of the lectures.

In line with previous findings, it was decided to make sure the live coding session is interactive. This would make lecture 2.2 more of a tutorial rather than a lecture, keeping students attentive.

Assignment 1

Time-wise, participants thought 2 hours would be sufficient. One participant mentioned that it could be difficult to create a grading scheme for this assignment since students are free in deciding how they want to present their diagrams. However, another participant noted that students are very likely to recreate given examples. It is expected that students will produce at least somewhat comparable results. Additionally, all students will have received the same lectures up to this point, so a general level of knowledge can be determined. Combined, it can be confidently said that it is possible to create a general grading scheme.

Assignment 2

The general consensus was that no final approval is needed for this assignment since students will be handing out their code to other students to receive feedback. As for the allocated amount of time, some participants thought it might even be too much time. However, the participants agreed that there might be some students who are really out of the loop with programming and need more time. This also showed up in the student survey. Conclusively, the amount of time given to the second assignment will stay the same.

Assignment 3

The participants noted that they thought that providing feedback on other students' code is a great learning experience since it can show the importance of writing clear code. As with the second assignment, some participants mentioned the amount of time allocated to this assignment might even be too much. However, it is always good to keep outliers in mind. Additionally, the third assignment is the last activity students have to do during the workshop, so if any students are done quickly they can just leave.

Additional workload

The TAs that were present mentioned they did not think that it would be too difficult for TAs to get into this subject, even if it is not currently in the curriculum. One participant mentioned it would be very useful for TAs to get to know more about this subject since it can help TAs with being able to explain certain programming problems more clearly.

Most participants agreed that grading the workshop would be difficult in general, since the submissions will cover a wide range. One participant suggested giving students who attend the workshop a pass/fail grade, to avoid having to create a grading scheme. However, another participant mentioned this might cause students to lose motivation since they do not get a grade. Another suggestion was to make the workshop pass/fail, but still give a grade for the third assignment. A grade would serve as an affirmation that the teaching staff has looked at their work. One participant mentioned a ranking system could be integrated with this idea, where students can see how well they performed compared to their peers. The way the workshop is graded is something to look into.

Lectures

In general, all participants agreed with the topics covered in the workshop and their allocated time. Only the live coding session might need some more time since there is an interactive part, as well as that it is expected that students have quite some questions. One participant mentioned that the introduction to Git lecture might be very dry and that students could have a hard time paying attention. It was suggested to integrate the lecture on Git with the live coding session and make it more tutorial-like. This would keep students engaged and make sure everybody is on the same page. However, this would mean even more time is needed for the live coding session since the Git lecture will be integrated and because it is expected that several students will not be able to follow along with the Git tutorial in one go.

9. Iteration 6: Final prototype

After discussing the ideas for the introductory workshop with students, alumni, TAs, and teachers, the actual content of the workshop was created. This chapter describes all the presentations and their content including in the software design workshop.

9.1 Software Design workshop

The final prototype is based on the final manual described in *Chapter 6: Iteration 3: Workshop manual* and the presentations made for the evaluation workshop. Additionally, the comments from the interviews and survey have been processed in this sixth iteration.

9.1.1 Layout software design workshop

The workshop has been designed as a separate teaching unit, with lectures and assignments separate from the existing curriculum. This means there is no need to integrate the workshop with certain subjects or modules. Based on the student survey found in Appendix 3, it was decided to place the workshop in module 3. Where exactly in module 3 the workshop would fit best still needs to be discussed with the module coordinator of module 3. The workshop spans two full workdays. The lectures have been designed to fit the time students can pay attention, and are alternated with assignments. It is important to allow students to put theory to practice. Additionally, as Anderson [44, 45, 46] describes, assignments allow the students to learn by doing. The schedule of the workshop can be found on pages 13-14 of the manual in Appendix 4.

9.1.2 Content software design workshop

All the presentations and their content can be found in Appendix 8.

9.1.2.1 Lecture 1.1: Introduction to software design

It was decided to kick off the workshop with a general **introduction to software design**. This would give participants the necessary knowledge to understand the assignments, as well as an idea about why software design is important. The presentation covers 3 different topics:

- What is software design?
- Why is software design important (for a CreaTe student)?
- Which tools are available to work with software design?
- Real-life examples of software design
 - UML
 - Class diagram
 - Interaction diagram
 - Use case diagram

9.1.2.2 Lecture 1.2: Explanation of assignment 1

After the introduction, the first assignment is introduced in the **Assignment 1** presentation.

9.1.2.3 Assignment 1

For the first assignment, students are given a textual description of what a piece of software should look like and need to create a design based on that. This design should include three diagrams: a class diagram, an interaction diagram, and a use case diagram. To help the participants a little, example designs for different textual descriptions are included in the explanation presentation. Two versions of this assignment will be handed out: half of the students will receive version A, while the other half will receive version B. The need for this is explained in subchapter 9.1.2.9 Assignment 3.

The design needs to be approved by a TA before the student can continue. This makes sure that students have seen what a correct diagram looks like and that students have a solid foundation to start with assignment 2. Additionally, as Winslow [29] states, it is best to make sure students show signs of clear understanding before introducing new topics. Robins et al. [28] also mention that, while models of a program can be beneficial for the understanding of a student, it is never certain if these models correctly represent the structure of the program without checking this first. So, TAs must check the diagrams of students before they continue to the next assignment.

9.1.2.4 Lecture 2.1: Explanation of assignments 2 and 3

After the first assignment, both the **second and third assignments** are explained. It was decided to explain these assignments together since students must know their code will be given to other students before they start writing it.

9.1.2.5 Lecture 2.2: Good coding habits and common mistakes

Since the workshop aims to bring students in contact with real-life practices, a lecture on **good coding habits and common mistakes** was added. Students will be given tips on how to keep their code clean and understandable. These tips and examples will help students with structuring their own code and making sure the code will be understandable for others. Additionally, it will help students with building up the skill of critically analyzing a piece of code and recognizing beginner mistakes.

9.1.2.6 Live coding session

Right after lecture 2.2, theory is put to practice in a **live coding session**. A teacher will show how they write code, from beginning to end. The teacher will describe their thought process out loud, so students can follow their train of thought. This is important since students often do not have the necessary strategies ready when working on code [10] and explicitly mentioning why a teacher does certain things can help with this. The session should have a clear structure: there should be a clear goal that the teacher is working towards, so students know what to expect. The sketch itself does not need to be complicated since simple examples are easier to understand. It is important to guide students from beginning to end. As Soloway [10], De Raadt et al. [12], and Robins et al. [28] state, one of the main struggles of students is that they do not know where to

begin given a problem statement. It is up to the teacher to decide how exactly the coding session will be shaped, but in Appendix 8 a list of necessary concepts to include is given. This coding session takes place using the Processing IDE, to accommodate the rest of the workshop. To make sure students can revisit the concepts, the coding session is recorded.

9.1.2.7 Lecture 2.3: Introduction to Git

Another real-life practice that is introduced is the usage of Git. Since Git is often used in some way or another by companies, it is beneficial for students to get in contact with it early in the curriculum. It will provide them with a way of working together on code that is very similar to what is being used in real life. For this lecture, SNT's GitLab will be used. Every student has free access to this as long as they are a student, so their repositories can be used during their whole student career.

9.1.2.8 Assignment 2

For the second assignment, students need to write code based on the design they made for the first assignment. Students should use the practices and tips that were explained to them during the previous lectures. As Soloway and Spohrer [30] state, it is important that programming and software design are treated separately. Both are essential to software development, but each has its own purpose. By having the second assignment build upon the first one, both topics can be treated separately while still keeping the assignments coherent.

9.1.2.9 Assignment 3

The code students write for the second assignment will be divided among all the students. Students who worked on version A for the first and second assignments will receive code from a team who worked on version B. This will ensure that the students do not need to work with the same concept during the whole workshop. Once students have received code, they need to provide appropriate feedback and again create three diagrams, one class diagram, one interaction diagram, and one use case diagram. This assignment is beneficial for students since it teaches them skills on how to read a program in addition to the skill of how to write a program taught in assignment 2. According to Winslow [29], this distinction is very important. Additionally, the repetition of having to create diagrams again can help with understanding software design concepts. As Yeh [1] found in his research, students become better at understanding and explaining programming concepts if they perform tasks repetitively.

The feedback will be both verbally and textually communicated to the students who originally wrote the code. This is beneficial for both the feedback-giving as well as the receiving team. As Soloway and Spohrer [30] state, students tend to interpret code in the way they understand it themselves, rather than what the code actually does. By having students state their interpretation of the code to the receiving team, a discussion can be started on the clarity of the code. The receiving team will know whether their work is understandable and if their intentions are clear, and the feedback-giving team trains their analytical reading skills and ability to give constructive feedback. After the students have completed the third assignment, the workshop is over.

9.1.3 Grading

The grading of the workshop is based on the third assignment only. The first and second assignments serve as an introduction to the topic. In the third assignment, all aspects covered by the workshop come together, creating a great opportunity for grading. Additionally, grading all three assignments would take quite some time for the teaching staff. While giving grades to the first and second assignments could be a motivating factor for students, this does not weigh up to the time needed to do all the grading.

It should be noted that students should only be graded based on the diagrams and feedback they provided for the third assignment. The grade should not be dependent on the code the students received since they were not involved in the process of writing it. This does mean that certain students will have to dig deep to find points of feedback, especially if they are not as good at programming themselves, while others can find points of improvement easily. However, whether the received code is of quality or not, students can still display skill in critically analyzing code and constructively providing feedback.

10. Conclusion

The goal of this project was to examine the curriculum of Creative Technology and explore the possibilities and effects of including more topics on software design. To guide the process of obtaining the necessary information, two research questions with several sub-questions were defined. The chapter includes the found answers to these questions and further describes the conclusions that could be drawn from this project. The answers to the sub-questions of both research questions are included in the respective discussions of each question.

10.1 Research questions

Before discussing the answers to the research questions, it is important to take the limiting factors of this project into account. Most obvious, COVID-19 has played a role in what could be accomplished. Due to the strict order to work from home when possible, all interviews, the survey, and evaluation sessions were conducted in an online environment. It is quite likely that this has influenced the number of people willing to participate in the evaluation sessions and the survey since they could not be promoted in person. It is unclear what the exact impact of this has been on the results. It could be that participants of the evaluation sessions were more likely to hand in digital diagrams instead of physical ones since they were already behind a computer. However, this is no different from the real workshop. Additionally, the online environment could have had an impact on what interviewees and evaluation workshop participants had to say. In a physical meeting, different answers might have been given and the dynamic between participants might have been different, leading to different results. Another limitation is that the planned workshop takes place during module 3. Since graduation semesters generally span module 1 and 2 or module 3 and 4, it is very hard for someone who is graduating to host the workshop as part of their graduation. This is also why the workshop has not been tested in its intended environment, but rather evaluation workshops were set up.

- **RQ 1:** In which parts of the CreaTe curriculum is Software Design currently present?

To answer this question, several interviews were conducted and a survey was sent out to all CreaTe students and alumni. Teachers agree that CreaTe could use more courses that include software design within the curriculum since it can help students with their programming skills. However, it is also noted that there is a smaller need for software design within a study such as CreaTe than for example in Computer Science. Alumni are in line with this opinion. While CreaTe did provide them with certain skills that are useful at work, some missing aspects were mentioned often. The main point mentioned often is that CreaTe should include more practices used in real life. An example of this is that many companies use Scrum and Agile methods, which are currently not included in the curriculum. Many interviewed alumni agreed that CreaTe could also benefit from adding more software design topics to the curriculum to better prepare future graduates. The answers from the student survey reflect this. Only 14 participants (28.9% of all participants) stated that they feel they know enough about software design. 7 participants (14.6%) noted that they learned software design and other programming skills from

following Computer Science modules. Around 35 participants (74.5%) mentioned that they think CreaTe should include more software design related topics in the curriculum.

- **RQ 2:** How can the CreaTe curriculum best be changed regarding Software Design?

A perfect answer for this question does not exist. However, based on the interviews held and the survey responses, certain decisions could be made. Module 3 was chosen as the optimal place to include software design. This was mostly based on the survey. Module 3 and module 7 both were mentioned 13 times. However, it became clear from the interviews with teachers and alumni that software design would be most beneficial to the curriculum when treated early on, so module 3 was chosen. To accommodate teaching students about software design, it was decided to design a workshop. The manual of the workshop can be found in Appendix 4, and the content of the workshop in Appendix 8. The requirements that were elicited from the alumni and teacher interviews and the student survey served a big role in determining what should be in this workshop. The following requirements were determined:

The project should (in no particular order):

1. Address **novice software design concepts**
2. Focus on **real-life practices**
3. Make sure students produce code that is **understandable** for others
4. Be **integrated with programming**
5. Include topics that **fit** within the current CreaTe program
6. Not take up too much **time** from the current curriculum
7. Be fit for students from **different backgrounds**
8. Preferably take place in **module 3**
9. Help students become **confident in their programming skills**
10. Include a relatively **big project**
11. Focus on **language fundamentals** and **problem-solving skills**

Most of these requirements are included in the different lectures and assignments of the workshop. Sadly, it was not possible to test the workshop in the intended environment, so it cannot be determined accurately whether the workshop addresses all requirements. For example, the inclusion of requirements 3, 6, and 9 can only be determined by hosting the actual workshop and evaluating the results. However, the workshop was specifically designed to include the requirements, which can be noted in the manual. Especially requirements 2 and 3 are included explicitly in the assignments.

Additionally, a goal was set up to make sure the project was fun for students next to being educational. While the actual workshop could not be tested in the intended environment, students were enthusiastic about the evaluation workshops that were set up. The results from these evaluations are described in *Chapter 7: Iteration 4: Evaluation workshops*.

10.2 Reflection on design process

An iterative design process was used during this project. This process was beneficial for the creation of the workshop and has helped with efficiently making progress. Every set of interviews led to a new design, as well as every evaluation session. By tackling the creation of the design in separate steps or iterations, the progress that was made every time can be clearly seen and reflected on. Additionally, by working on the workshop step by step, documenting the progress became easier. Results from interviews, the survey, and evaluation sessions could immediately be implemented in the design, and the differences documented. While the online environment had its negative impact on the design process, some benefits could also be seen when hosting the interviews. It became clear that people have become accustomed to working in an online environment and planning in online meetings, also with people who do not live nearby, was quite easy. Additionally, the evaluation workshops were also easier to plan digitally than it would have been to plan them physically. Looking back, the benefits of hosting the interviews and evaluation sessions online might even outweigh the benefits of doing them physically anyways, even if the possibility had been there.

It is a shame that the actual workshop could not be tested during the timeframe of this graduation project. However, the evaluation sessions still have provided beneficial insights into what students think of the way software design is introduced and whether they think it is a helpful skill. It has become clear that many different parties think that CreaTe could benefit from more software design related topics and that providing basic knowledge on the topic in a workshop is a good way to present the information. Nevertheless, more is needed for the evaluation sessions to determine if the designed workshop can be implemented in the curriculum. It is recommended to host the workshop in its intended environment first and make sure the programme staff approves of the design. This and further recommendations are discussed in the next chapter, *Chapter 11: Future work*.

11. Future Work

11.1 Workshop evaluation

To evaluate whether the workshop is beneficial to the curriculum of CreaTe, it needs to be tested in its intended environment. This means that the workshop should be hosted during module 3 according to the manual. During the workshop, the following aspects need to be assessed:

- The difficulty level of the assignments
It is important to know whether the assignments that were created for the workshop are of adequate level, taking the knowledge students should have in module 3 into account. Additionally, it should be evaluated whether the amount of time that students have per assignment is in line with the difficulty level.
- The understandability of the assignments
The assignments should not be wrongly or differently interpreted by students, so it should be determined whether all students understand the assignments in the same way.
- The effects of the lectures on student knowledge and motivation
Do the lectures provide enough knowledge for students to finish the assignments? Is the content of the lectures presented interestingly and are students able to keep their attention? Should the lectures take less time or are more breaks between lectures needed? These questions should be answered during the evaluation of the workshop.

It is also possible to host the workshop with TAs first. For example, a test workshop could be hosted in the first module where TAs join as participants. This would make sure that some TAs already receive the necessary knowledge to help during the actual workshop and that the workshop content can be evaluated before taking place in module 3.

Additionally, it could be worthwhile to investigate the long-term impact of the workshop. By hosting the workshop over multiple years, the knowledge and skills of students when it comes to software design and programming can be assessed.

11.2 Workshop

Some possible changes came up during the final stages of this graduation project. It was mentioned that the lecture on good coding habits could be interwoven with the live coding session, instead of first hosting the lecture and then the session afterward. This could help with keeping students engaged and gives possibilities for more interactivity. Additionally, the lecture on Git needs to be made student-proof. This means that more information needs to be included in the workshop manual on what to do if a student gets stuck using Git. Some kind of *Frequently Asked Questions* section, which is also shared with the students, could be very beneficial. Next to that, it could be considered to create separate assignment and lecture documents, instead of

having a manual with examples interwoven in it. Having separate documents per lecture and assignment, or per day, or per topic, could give teachers (or any other beneficiaries) a better view of the workshop content. Lastly, the workshop manual should be updated according to the results of hosting the workshop.

11.3 Involvement programme staff

Due to time constraints, some teachers and the module coordinator of module 3 could not directly be involved with the creation of the workshop. Since their opinions are very valuable to the project, it is strongly advised to gather their opinions on the workshop. Additionally, the Programme Committee of CreaTe needs to agree with the content of the workshop. They have a clear view of whether the workshop is in line with the goals of CreaTe and the University.

11.4 Connection to other modules

In theory, the workshop could take place anywhere within the curriculum since it has been designed to be a separate unit. A connection between software design and programming is reached within the workshop itself. However, this does not mean that the content of the workshop needs to be contained to the workshop only. It is worthwhile to investigate whether certain aspects of the workshop could be integrated with other modules as well. The diagrams that are discussed within the workshop could already be treated in the first or second module, for example. This would help students with understanding the basic theory of programming already at the beginning of their programming career. Additionally, it could be investigated whether the workshop could be integrated with Module 3. The workshop focuses on real-life practices, which could be connected to the focus on companies in the third module. Students could be asked to create a (mock-up) piece of software, including proper software design, as if they would present it to a company. This would introduce students to what it is like to present software to an external party. In the fourth module, students have to complete a large programming project. The workshop could help students with preparing for this project, especially if the project would explicitly include software design. Students could be asked to create software design diagrams to prepare for the realization of their project. This also is the case for the sixth module, where students also have to complete a large programming project. Module 7 could serve a similar purpose as Module 3 would serve to Module 4: during the seventh module, students could be given software design related assignments to prepare for the project in Module 8, which would include an obligatory software design part. Module 7 also focuses on companies, so the real-life aspect could also be taken into account here. All these examples can be considered and would need to be discussed in-depth with the relevant teaching staff.

Appendices

Appendix 1: Interview summaries

1.1: Teaching staff interviews

Interview 1

- There's an obvious synergy between software design and code quality: it helps to create a mental model of the project
- Explicitly teaches good coding habits
- Doing design makes projects understandable for others. Sometimes, UML is useful here
- Design is about building abstractions and needed to combat complexity
- UX for example would be an interesting design topic for CreaTe

Interview 2

- There is less software design present in CreaTe in comparison to TCS at the moment, but also less necessary. (for example, UML is not needed)
- There is a big gap between m1 and m4 programming-wise
- More design in the first year could perhaps help, but it could also be very dry material that's hard to swallow for lots of students, so you will really need to sell it
- In M6, there is a lot of space to improve things. Students should know more about design than they do at this point. Some kind of road map would be nice to have, design-wise
- The interviewee would say: design first, then program. It plays more to the strength of designing
- The integration between software design and programming is important

Interview 3

- A bottom-up approach is preferred: let students come across problems and then explain why they happen and how to solve them. If you get a recipe given to you, there is little room for creativity
- The programming level is very much functional, even in M5 ST.
- In M8, there is a focus on project work, which is good! However, students are not required to show their progress in teamwork skills
- UML could be used in CreaTe, but just for the concepts it shows: teaching UML without using UML
- It's important to take different backgrounds of students into account
- Software Design is super relevant for CreaTe, but the importance of the topic should also emerge from the students themselves. The study should not give a how-to method, and the importance should be stressed.

Interview 4

- Design could be added to CreaTe. However, there is no need to be able to go into depth on design topics since this is not the focus of CreaTe. The main goal is to be able to communicate with other people who have a different work field
- There is not a lot of time to spend on design within CreaTe. However, perhaps some time should be made free for the topic

1.2: Alumni interviews

Interview 1

New

- Add scrum/agile, DevOps, Continuous integration
- Focus more on going into specific methods real engineers use, and explain why they are popular
- In general: focus more on real-life practices of working in a company

Keep

- Learning how to get knowledge yourself
- Teaching students how to be the translator of different fields
- Working together on a project and having specific roles

Interview 2

New

- Some coding aspects
- The technical aspects of a system (for example, the split between front- and backend)
- Add scrum/agile
- The steps after writing working code: testing and release
- Working together on code
- In general: focus on more real-life practices of working in a company

Keep

- Thinking in smaller steps: there is a problem, we need a solution, what are the options?
- Thinking broadly and out-of-the-box
- Project work, being comfortable with working in a team

Interview 3

New

- A Bigger focus on data protection and privacy and the ethics behind this
- Scrum/agile
- GitHub
- IDEs
- Different programming languages (that are less outdated and heavy), like Python instead of Java
- UML diagrams!
 - Not so much the professionalism, but more the thinking processes behind creating such a diagram
- In general: focus on more real-life practices of working in a company

Keep

- Being able to visualize your ideas
- Learning how to find knowledge yourself

Interview 4

New

- Focus on students being able to deliver code that is understandable and workable for others
- Take a look at the goals of CreaTe. Right now, there is a big focus on being able to prototype.

- Students are also not expected to go into the depth of projects. The Interviewee has learned the most from teaching it to himself in personal projects.
 - What is in line with this, is that students are not expected to write beautiful code, just working code.
- Eliciting requirements in combination with creating a suitable piece of software for them
- A bigger focus on design, but taking different student background into account
- More targeted feedback from TAs who know what they are doing

Keep

- The philosophy behind CreaTe: being able to learn things yourself, talking to different kinds of people, understanding what people mean, not being afraid to confront others: tinkering and the freedom to just mess around
- Open assignments and the openness for personal projects

Appendix 2: First draft introductory workshop manual

Dag 1

(9:00 - 10:30) De eerste dag zal beginnen met een korte intro over wat software design is en waarom het belangrijk is. Ook wordt de eerste opdracht ingeluid en voorbeelden gegeven.

(10:30 - 12:30) Daarna gaat men in groepjes van twee aan de slag met een design maken, door middel van zinnen te ontleden. Zelfstandig naamwoorden worden classes, werkwoorden worden functies, bijvoeglijk naamwoorden worden attributen, etc. Dit design wordt gecontroleerd en beoordeeld door TAs.

(13:45 - 15:00) Zodra het design is goedgekeurd, is het de bedoeling dat er aan de hand van dat design code wordt geschreven. Eerst wordt een korte presentatie gegeven over veelgemaakte fouten als het komt tot code schrijven. Het liefst is dit een live coding sessie, waarbij de docent hardop vertelt hoe hij/zij het aan zou pakken om van een design naar een programma te gaan en daarbij live code schrijft. Veelgemaakte fouten worden benadrukt. Aan het einde wordt een korte introductie gegeven over opdrachten 2 en 3, benadrukt dat code voor andere begrijpelijk moet zijn, en een korte introductie gegeven over GitHub.

(15:00 - 17:30) De studenten gaan aan de slag met coderen. Hiervoor wordt GitHub aangeraden. Als studenten daar nog moeite mee hebben, kunnen TAs helpen. De deadline voor code inleveren staat op 22:00.

Dag 2

Voor 9u moet een schema worden gemaakt voor welke teams aan elkaar hun code gaan geven.

(9:00 - 9:30) Code wordt overhandigd (geen uitleg - de code moet vanzelfsprekend zijn). Eventuele problemen daarmee worden opgelost.

(9:30 - 12:00) Studenten hebben de tijd om van de programma's een design te maken, en eventuele aanbevelingen te doen aan het team. Dit design moet weer worden goedgekeurd door een TA.

(12:00 - 12:30) Uitlooptijd en afsluiting.

Appendix 3: Student survey summary

Demographics

In total, 48 people participated in this survey. However, one response was found to be duplicate and was removed, leaving a total of 47 responses.

9 participants (19.1%) currently follow a module in the first year of CreaTe, who all started their studies in the academic year 2020 - 2021. These participants will be referred to as 'first-years'.

9 participants (19.1%) currently follow a module in the second year of CreaTe, of which 7 participants started their studies in the academic year 2019 - 2020 and 2 in 2018 - 2019. These participants will be referred to as 'second-years'.

20 (42.6%) currently follow a module in the third year of CreaTe, of which 15 started their studies in the academic year 2018 - 2019 and 5 in 2017 - 2018. These participants will be referred to as 'third-years'.

9 participants (19.1%) already have their Creative Technology diploma, of which 2 students started their studies in the academic year 2017 - 2018, 4 in 2016-2017, and 3 before 2016. These participants will be referred to as 'alumni'.

Q1: During which Creative Technology module did you learn most about programming?

Module	Participant total	Demographics	
1	10	6 first-years 2 third-years	2 second-years 0 alumni
2	5	2 first-years 1 third year	0 second-years 2 alumni
3	2	1 first year 1 third year	0 second-years 0 alumni
4	10	0 first-years 5 third-years	4 second-years 1 alumnus
5: Interactive Media	3	0 first-years 2 third-years	0 second-years 1 alumnus
5: Smart Technology	1	0 first-years 1 third year	0 second-years 0 alumni

6	11	0 first-years 7 third-years	2 second-years 2 alumni
7	1	6 first-years 2 third-years	2 second-years 0 alumni
8	0	0 first-years 0 third-years	0 second-years 0 alumni
11	0	0 first-years 0 third-years	0 second-years 0 alumni
12	0	0 first-years 0 third-years	0 second-years 0 alumni
I am not sure	4	0 first-years 1 third-years	1 second year 2 alumni

Q2: Do you feel confident in your programming skills? (rate 1 - 5)

Rate	Participant total	Demographics	
1	4	0 first-years 3 third-years	0 second-years 1 alumnus
2	11	2 first-years 4 third-years	2 second-years 3 alumni
3	13	5 first-years 2 third-years	3 second-years 3 alumni
4	14	2 first-years 7 third-years	4 second-years 1 alumnus
5	5	0 first-years 4 third-years	0 second-years 1 alumnus

Average first-years:	$((1 \times 0) + (2 \times 2) + (3 \times 5) + (4 \times 2) + (5 \times 0))/9$	= 3
Average second-years:	$((1 \times 0) + (2 \times 3) + (3 \times 3) + (4 \times 4) + (5 \times 0))/9$	= 3.44
Average third-years:	$((1 \times 3) + (2 \times 4) + (3 \times 2) + (4 \times 7) + (5 \times 4))/20$	= 3.25
Average alumni:	$((1 \times 1) + (2 \times 3) + (3 \times 3) + (4 \times 1) + (5 \times 1))/9$	= 2.78
Average total:	$((1 \times 4) + (2 \times 11) + (3 \times 13) + (4 \times 14) + (5 \times 5))/47$	= 3.11

Q3: To what degree did the Creative Technology program assist you in becoming confident in your programming skills? (rate 1 - 5)

Rate	Participant total	Demographics	
1	2	0 first-years 1 third year	0 second-years 1 alumnus
2	12	0 first-years 10 third-years	0 second-years 2 alumni
3	19	5 first-years 4 third-years	5 second-years 5 alumni
4	10	2 first-years 5 third-years	3 second-years 0 alumni
5	4	2 first-years 0 third-years	1 second year 1 alumnus

Average first-years: $((1 \times 0) + (2 \times 0) + (3 \times 5) + (4 \times 2) + (5 \times 2)) / 9 = 3.67$

Average second-years: $((1 \times 0) + (2 \times 0) + (3 \times 5) + (4 \times 3) + (5 \times 1)) / 9 = 3.56$

Average third-years: $((1 \times 1) + (2 \times 10) + (3 \times 4) + (4 \times 5) + (5 \times 0)) / 20 = 2.65$

Average alumni: $((1 \times 1) + (2 \times 2) + (3 \times 5) + (4 \times 0) + (5 \times 1)) / 9 = 2.78$

Average total: $((1 \times 2) + (2 \times 12) + (3 \times 19) + (4 \times 10) + (5 \times 4)) / 47 = 3.04$

Q4: To what degree does your confidence in your programming skills influence your motivation to write code? (rate 1 - 5)

Rate	Participant total	Demographics	
1	0	0 first-years 0 third-years	0 second-years 0 alumni
2	4	1 first year 1 third year	1 second year 1 alumnus
3	6	2 first-years 2 third-years	0 second-years 2 alumni
4	15	3 first-years	4 second-years

		6 third-years	2 alumni
5	22	3 first-years 11 third-years	4 second-years 4 alumni

Average first-years:	$((1 \times 0) + (2 \times 1) + (3 \times 2) + (4 \times 3) + (5 \times 3))/9$	= 3.89
Average second-years:	$((1 \times 0) + (2 \times 1) + (3 \times 0) + (4 \times 4) + (5 \times 4))/9$	= 4.22
Average third-years:	$((1 \times 0) + (2 \times 1) + (3 \times 2) + (4 \times 6) + (5 \times 11))/20$	= 4.35
Average alumni:	$((1 \times 0) + (2 \times 1) + (3 \times 2) + (4 \times 2) + (5 \times 4))/9$	= 4.00
Average total:	$((1 \times 0) + (2 \times 4) + (3 \times 6) + (4 \times 15) + (5 \times 22))/47$	= 4.17

Q5: Which factors of Creative Technology had the greatest impact on your confidence in your programming skills?

1 participant mentioned grades are important.

2 participants mentioned that the fact that the programming courses and weekly assignments are mandatory helped.

3 participants mentioned that the subjects made a difference: Algorithms, AI theory, and the Programming sports day.

2 participants mentioned Python, a language that they did not use before.

1 participant mentioned the focus on different programming languages.

3 participants mentioned the process of getting or not getting the help you need from TAs and teachers.

2 participants mentioned individual assignments.

9 participants mentioned that they learned from (completing) the end projects and bigger coding projects like the module 4 and module 6 programming projects.

2 participants mentioned that they learned a lot from working on their own projects, outside of CreaTe.

3 participants mentioned that they liked the visual feedback that is given by Unity, Processing, and building things with Arduino, so you can really see what you have programmed. 1 participant specifically mentioned modules where you can program a game as having an impact on programming skills.

1 participant mentioned they liked the feedback and interactions with Angelika in module 4.

1 participant mentioned they learned a lot from helping others.

5 participants mentioned that practicing a lot is the most important thing and that reaching increasingly difficult goals feels very motivating.

1 participant mentioned they were motivated by seeing less knowledgeable younger students.

1 participant mentioned they learned all their programming knowledge from CreaTe since they had zero experience.

1 participant mentioned they often took on roles that did not contain programming in group projects since they believe that others were better at programming.

2 participants mentioned they liked the hands-on approach in modules 1 and 2, and that they were disappointed that this stopped afterward.

2 participants mentioned that, even though they were able to keep up with all assignments, the oral examination was more strict than they expected.

1 participant mentioned they got demotivated from seeing their peers make more advanced things.

2 participants mentioned they thought the first module went too fast. This made them feel alone and behind, making it scary to ask questions.

1 participant mentioned they thought that programming is not being taught properly within CreaTe.

2 participants replied something along the lines of “none”.

2 participants replied that they were not sure.

Q6: Are there other factors (outside of Creative Technology) that influenced your confidence in your programming skills?

3 participants mentioned they learned a lot from (computer science) acquaintances since they were able to explain code and programming faster than TAs and teachers. Additionally, having peers help out formed a lower threshold than for asking TAs or a teacher since it felt less like asking stupid questions.

7 participants mentioned that they did modules of Technical Computer Science, which had a big influence on their programming skills.

4 participants mentioned game development in their free time.

4 participants mentioned watching YouTube videos in their free time.

4 participants mentioned being a TA.

11 participants mentioned doing personal (programming) projects.

3 participants mentioned they learned some programming in high school.

1 participant mentioned that they did an online programming course.

3 participants mentioned game jams.

1 participant mentioned finding correct solutions quickly.

1 participant mentioned their dad could help them out since he codes for a living.

1 participant mentioned Daniel Shiffman.

2 participants mentioned they got motivated from seeing others having less programming knowledge.

1 participant mentioned that real-life programming is very different from CreaTe programming.

1 participant answered that they were not sure.

2 participants answered “no”.

11 participants left this answer blank.

Q7: Do you feel as if there are any modules that currently lack education on programming? If so, which one(s)?

20 participants mentioned that modules that form a gap in programming education should integrate more programming courses. Module 3 and module 7 are mentioned often, also because participants feel that these modules have enough room to fit more topics. Module 1, 2, 4, 5, and 8 are also mentioned, but less frequently.

1 participant mentioned they would like to see a bigger focus on programming languages that are used in the real world regularly instead of Processing.

1 participant mentioned they did not like that the type of programming between modules is not consistent.

1 participant mentioned they did not like the interactive video framework in module 1.

1 participant mentioned that module 2 could focus more on actually programming the smart environment you're making.

1 participant mentioned they found module 2 hard to follow since only videos of unpublished example code were given.

1 participant mentioned that the programming in module 2 would be better off in another module.

1 participant mentioned that they did not feel like they learned a lot of programming in module 4 since they did not feel they could write better programs at the end.

2 participants mentioned they think module 6 lacks depth when it comes to programming.

1 participant mentioned they think module 6 focuses on the wrong things. They would rather see a bigger focus on learning how to design a program and interaction between classes instead of complexity and AI.

1 participant mentioned that they think module 6 has a bad introduction/base explanation and that it feels like you are thrown in the deep since you start with a new programming language, and therefore seems intimidating.

1 participant mentioned that there is a lack of Unity coding in modules 5 and 8 because the project had to be made in VR due to COVID-19.

1 participant mentioned that there should be a bigger focus on web development since this is a skill that can be applied to almost every company.

1 participant mentioned they did not like Web Technology, as it was not very well set up.

3 participants mentioned they think the amount of programming is good.

1 participant mentioned that no modules need more programming courses since they believe that not every module needs to contain it.

10 participants left this answer blank.

Q8: Are there any modules that already have too much programming in it? If so, which one(s)?

11 participants replied “no”.

1 participant mentioned that year 1 contains a lot of programming.

1 participant mentioned they found module 1 very challenging since they never programmed before.

1 participant mentioned they found the jump from module 1 to module 2 quite big.

1 participant mentioned they found module 2 challenging, but not too much.

2 participants mentioned that module 4 went very rapidly from topic to topic, making it quite hard considering the amount of programming knowledge up to that point.

2 participants mentioned that the step from Processing to Python in module 6 is a big step.

Not necessarily too much programming, but there is not enough time to do this well.

1 participant mentioned that module 6 has a lot of programming, but that a big part is optional.

1 participant mentioned that some programming in some modules definitely caused panic due to lack of time, but that that was mainly caused by having many deadlines around the programming deadline.

1 participant mentioned that there should be a bigger focus on “useful” programming which can be utilized in real life.

2 participants mentioned they like programming, so they prefer more over less

1 participant left this answer blank.

Q9: If you could change the topics on programming that are treated within the Creative Technology program, what would you change?

12 participants mentioned that there should be a bigger focus on language fundamentals (how does Java work, syntax, documentation, best practices) and problem-solving skills. To quote one answer: *“What I realised in M1 is they only gave like 2-3 weeks to actually learn the language and understand how programming even works and then we are already asked to*

make things move in Processing, but nobody even understood yet how programming even worked. Until now, most people don't understand the languages they work with (constraints, features, quirks), and/or don't know how to solve programming problems when they encounter one. [...] Too much focus on making fancy things by copy pasting this or that algorithm and tweaking some numbers instead of learning and understanding how to utilise a language to create what you want. ”

1 participant mentioned they think students should learn more about breakpoints for debugging purposes.

5 participants mentioned they really miss information on the design of software systems within CreaTe.

4 participants mentioned they think there should be a bigger focus on software engineering skills (knowing software lifecycles, using agile development using git, how to test your software).

1 participant mentioned that a trial-and-error method is not a good theoretical basis for programming and that the programming skills of CreaTe students could be improved by changing this method. This could also help with boosting student confidence.

1 participant mentioned it is very important for TAs to understand that just giving pieces of code does not help in understanding programming.

1 participant mentioned they think module 1 is packed with information if you are not familiar with programming at all.

1 participant mentioned they really disliked the math assignment during Interactive media in 2020-2021 since essential materials were delivered way too late.

1 participant mentioned they did not like the Programming and Physical Computing course in module 2 since they could not keep up and did not feel prepared for the final assignment.

1 participant mentioned they were not motivated to work on assignments since they felt useless, and would like programming courses to focus more on 'useful' things.

4 participants mentioned that they would like to see less visual programming (Processing) since this often does not translate well to programming jobs.

1 participant mentioned they would rather see a bigger focus on C and MicroPython.

1 participant mentioned that they would like to see more programming on applications (Android studio) and JavaScript coding.

2 participants mentioned a bigger focus on Web development.

1 participant mentioned the use of Raspberry Pis instead of Arduino.

6 participants mentioned that more Python should be included in the curriculum.

1 participant mentioned they would like to see programming courses for Excel.

1 participant mentioned CreaTe should focus more on programming models.

2 participants mentioned they would like the programming courses to also look at real-life codes and big programs.

2 participants mentioned they would like to include C++ since it seems useful.

1 participant mentioned they would like to see more creative projects.

1 participant mentioned they would like a bigger focus on UI and UX.

1 participant mentioned that it might be an idea to divide CreaTe students into a group of people that can program well and those who cannot really since the interests of students are diverse and those who are not that interested in programming quickly fall behind with programming.

1 participant mentioned they would make the programming courses fit better with prototyping skills.

2 participants replied “no”.

1 participant is not sure.

7 participants left this answer blank.

Q10: Do you have any other comments related to this topic you would like to share?

2 participants mentioned that CreaTe programming feels a bit like a crash course. While this is not a problem in itself, it does create an awkward divide between people who want to become better at programming and those who are not interested. The programming courses should have to cater to both.

1 participant mentioned that it is a real problem that quite some second- and third-year students severely lack programming skills.

1 participant mentioned that they feel as if you can graduate from CreaTe with almost no actual programming skills, so more and better programming courses are needed.

1 participant mentioned they think CreaTe teaches how to play with code rather than to actually code.

1 participant mentioned they think programming is taught as a tool instead of a way of thinking, which makes building projects from scratch very difficult/daunting.

1 participant mentioned that they think TAs have a very important role in explaining stuff.

2 participants mentioned that it is quite a problem that, if you work together on a programming project and one of the group members is not good at programming, it is very easy to just lift off of the person who is. To quote one answer: *"... I experienced that if you collaborate with someone who is also not good at programming, that the assignment will be hard to complete, but if you collab with someone who is better, you lose as well since you can only sit back and watch as the other does most of the work. And then there is the oral assessment... Collaborating with someone who is better at programming and then have an oral assessment is the worse.. I feel like I can't explain complex things, but the other can, so you easily give the idea that you did nothing..."*

1 participant mentioned they found it hard to find a partner with a similar skill in programming.

1 participant mentioned that it feels like the modules that focus on electrical engineering are more focused and taught better than the programming courses.

1 participant mentioned they like to switch to Python in module 6.

1 participant mentioned they learned the most from modules that included Python.

1 participant mentioned they think the courses related to programming are well taught with adequate support throughout the module.

1 participant mentioned that they like programming that is combined with making things move physically since it is memorable.

1 participant mentioned they think CreaTe offers a welcoming ambiance concerning learning how to code, which is very important, especially for students who do not have prior experience.

1 participant mentioned they quickly get annoyed when they don't know how something programming-related works for too long, and then quit.

1 participant mentioned that the end assignment grading of module 2 felt unfair and irritating.

1 participant mentioned they think Processing is useless and should be taken out of CreaTe.

1 participant mentioned a course at Linköping University, where students work on an existing open-source project of their choice in teams of 5-10 people. This helps students with learning how to code collaboratively using GitHub, work in a team and on large existing codebases.

2 participants suggest taking inspiration from the Design course in module 2 of Technical Computer Science.

1 participant mentioned that code explanations during lectures could be a good addition.

1 participant mentioned they would like a bigger focus on search algorithms and the creation of games that use that.

1 participant mentioned they would like a bigger focus on Unity.

1 participant mentioned that Angelika is great.

2 participants answered "no".

23 participants left this answer blank.

Q11: Do you feel as if you know enough about software design, taking your study career into account?

Answer	Participant total
Yes	14 (28.9%)
No	27 (57.4%)
Other	6 (16.7%)

2 participants mentioned only students who did a minor or pre-master in Computer Science know enough about software design within CreaTe.

1 participant mentioned they think they do know some things about the structure of code and basic class diagrams, but that they are not great at designing code or visualizing the designs.

1 participant mentioned they feel confident in their software design skills, but that they do not understand how they are taking their study career into account.

1 participant mentioned they know a bit, but are not sure how much there is to know about the topic.

1 participant mentioned they are not sure.

Q12: Do you think Creative Technology should include more software design related topics in the program? If so, where within the program should these topics be treated?

10 participants mentioned they think learning how to sketch out a program before starting would be a great addition.

2 participants mentioned they think software design could help with building confidence in programming skills.

1 participant mentioned they think that software design is more important than programming in a lot of cases.

4 participants mentioned that software design can help with understanding a system.

1 participant mentioned that software design can help with selecting the proper language to program in.

1 participant mentioned individual exercises are important since they require the student to understand the topic.

1 participant mentioned software design is important since it allows you to communicate ideas to people who know nothing of programming.

1 participant mentioned it would be great to focus on software design instead of on specific skills (examples mentioned: making a mandala using math formulas, the video project).

1 participant mentioned it is a good idea to make sure designs are handed in before writing code since otherwise, it is tempting to create the layout of a program after it is already done and working.

10 participants mentioned it would be good to add software design early in the curriculum.

1 participant mentioned basics could be added to module 2 and more details to later modules.

3 participants mentioned module 3 could use more software design since you get a use case for the first time.

2 participants mentioned module 4 could benefit from more software design since the programs get more complex.

1 participant mentioned it could be added to Interactive Media.

1 participant mentioned it is not necessary to go as in-depth as, for example, Technical Computer Science.

1 participant mentioned it should be mostly added in the back end since UI and UX already have enough presence.

1 participant mentioned it could be added to the Python course.

2 participants mentioned it could be added to all modules that include programming.

1 participant mentioned it could be added as an additional course or workshop within a module.

3 participants replied “yes” without an explanation.

2 participants replied “yes”, but were not sure where it could be added.

4 participants replied “no”.

1 participant replied “no”, with the explanation that the information that is already being taught should be explained differently so students can understand it better.

2 participants replied “no” since they do not like programming and feel as if there is already too much.

1 participant replied “no” since, while it is an important topic, students did fine without it.

3 participants are not sure.

Q13: Do you have any other comments related to this topic you would like to share?

1 participant mentioned that students should deliver an initial design before they start working on their program. This is important for grading and the careers of the students later on.

1 participant mentioned that the focus should not be on UML diagrams, but more on the part of thinking about the structure of your code.

1 participant mentioned they think students did not learn a lot about software design, or at least were not taught it well.

1 participant mentioned that structure and its impact became clear in module 6.

1 participant mentioned they think no programming design is being taught in CreaTe.

1 participant suggested bringing back techno-dramas.

35 participants replied “no”.

Appendix 4: Final version of the introductory workshop manual

INTRODUCTION TO SOFTWARE DESIGN **Makeathon in Creative Technology Module 3** **Manual 2020-2021**

Written by Hannah Ottenschot
April, May, June 2021

Table of contents

Introduction	2
Assignments and lectures	3
Day 1	3
Lecture 1.1: Introduction to software design	3
Lecture 1.2: Introduction to assignment 1	3
Assignment 1	3
Lecture 2.1: Introduction to assignment 2 and 3	6
Lecture 2.2: Good coding habits and common mistakes	6
Lecture 2.3: Introduction to Git	6
Day 2	11
Assignment 3	11
Schedule	13
Role of Introduction to Software Design within CreaTe curriculum	15
3.1 Learning goals	15
3.2 Connection to module 3	16
3.3 Connection to other modules	16

Introduction

Based on research, it has been concluded that it could be beneficial to integrate more software design aspects within the Creative Technology (CreaTe) curriculum. For this reason, a two-day workshop for introducing the topic Software Design to CreaTe students has been designed. As of now, the proposal aims to add this workshop to module 3. In section 3.2 Connection to module 3, the reasoning behind this has been written down. The workshop aims to structurally introduce the topic of software design to CreaTe students. They will be asked to make designs themselves, and write code based on their own designs. It is believed that the ability to create and understand designs is a crucial skill for Creative Technologists and that the workshop could be beneficial for the overall understanding students have of programming. In section 3.1 Learning goals, the intended learning outcomes for students have been described. These learning goals will hopefully be reached by having students complete multiple assignments, introduced by several small lectures and one live coding session. The proposed assignments and lectures can be found in the Assignments and lectures section. Additionally, a schedule has been created for the workshop, which can be found in the Schedule section.

1. Assignments and lectures

The students will complete several assignments during the workshop. These assignments will be introduced and guided by different lectures. On the first day, there will be two lectures (1.1 and 1.2) along with the first assignment followed by three lectures (2.1, 2.2, and 2.3) and the second assignment. On the second day, students can continue working on the second assignment and finish with the third assignment.

Day 1

Lecture 1.1: Introduction to software design

Since this is one of the first times students come in contact with software design, a small introduction is given to explain what the topic is about and why it is important.

Topics that could be covered in this lecture:

- What is software design?
- Why is software design important (for a CreaTe student)?
- Which tools are available to work with software design?
- Real-life examples of software design
 - UML
 - Class diagram
 - Interaction diagram
 - Use case diagram

Lecture 1.2: Introduction to assignment 1

The first assignment is introduced. The concept is explained, and (simple) example outcomes are shown. Students are instructed to ask questions to TAs and have their design approved.

Assignment 1

For this first assignment, students are asked to create a design. This will not be done based on software, but a textual description. Students need to derive a design from a given text according to the nouns, verbs, and adjectives. The design should include a class diagram, an interaction diagram, and a use case diagram. The interaction part of the textual description is left vague on purpose, to make sure students think for themselves what possibilities they consider for an interactive system.

The goal of this assignment is to introduce students to eliciting requirements from a description, which is often necessary when it comes to programming, as well as to have them experiment with the possibilities that are out there to create designs and look at the possibilities from different angles. The textual descriptions and designs themselves don't have to be very elaborate yet.

Two versions (A and B) of this assignment will be available. Half of the student groups will receive version A, while the rest will receive version B. The difficulty of these assignments will be roughly the same, but the textual descriptions will differ. This split will be important for assignment 3.

A simple example:

A museum in Enschede has an exhibition coming up, which focuses on Spring. The exhibition is meant for children up to 12 years old. There will be several different interactive installations that the children can use, each showing a different event associated with Spring. It is your task to write Processing sketches that portray the events and include the necessary interactivity.

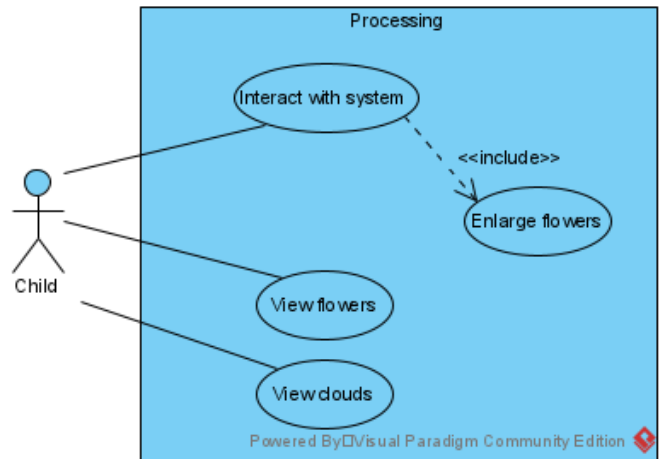
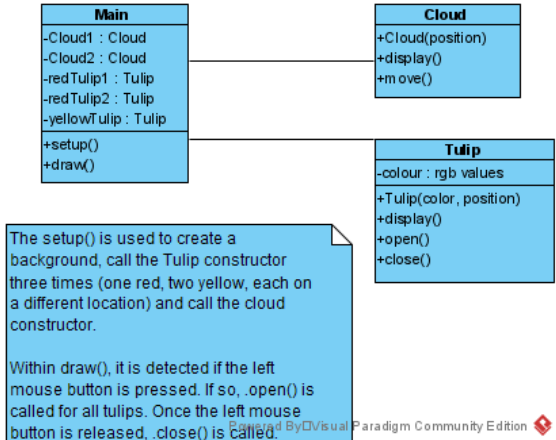
Sketch #1: Flowers in bloom

This sketch should portray flowers blooming. Three closed tulips are shown in different locations: one red, and two yellow ones. When a child interacts with the system, the flowers should open. Once the child stops interacting, the flowers should close again. The background of the sketch should show the sky with moving clouds and a green grass field.

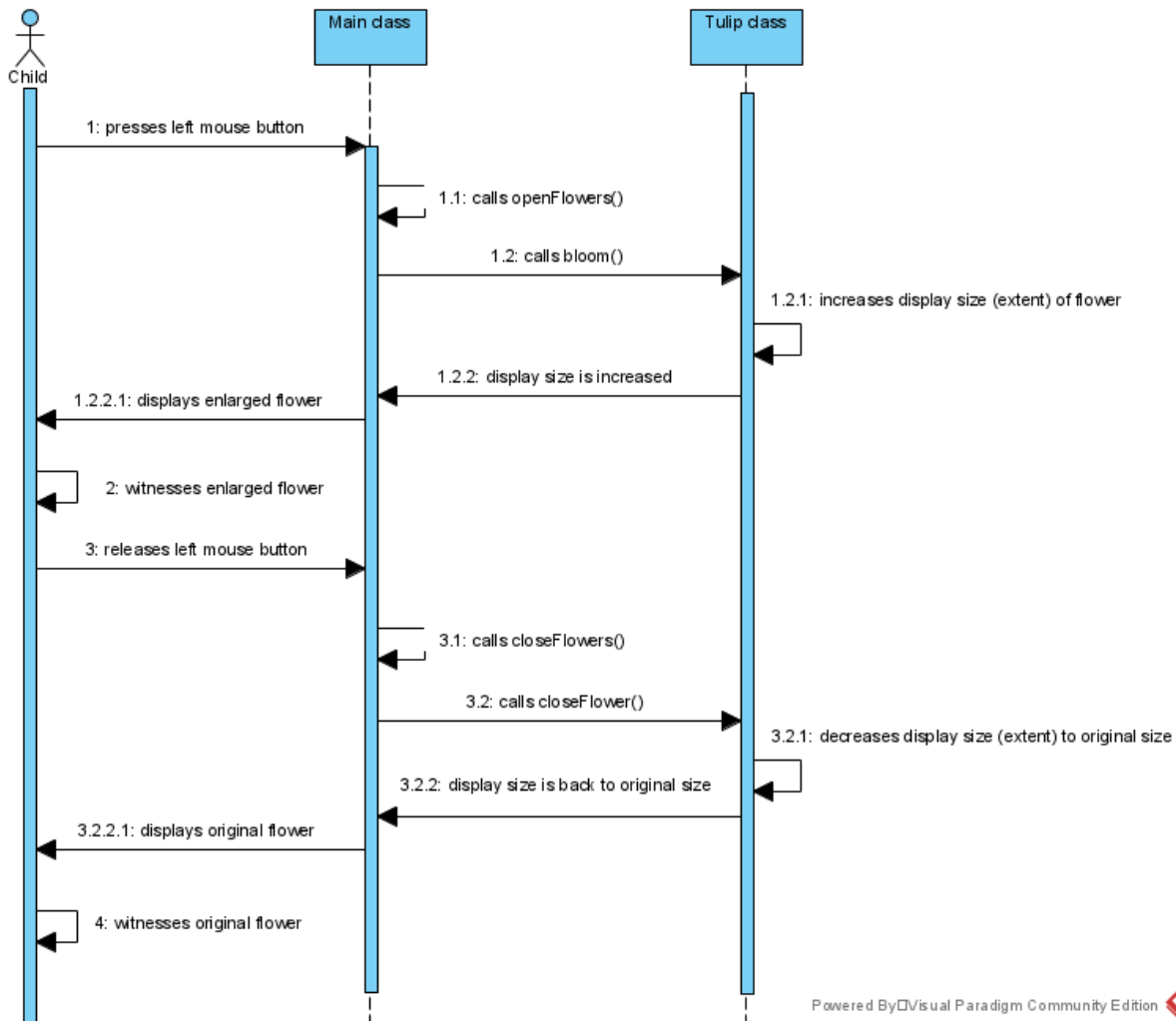
For this first sketch, students could elicit the following:

- A 'tulip' class is needed for the tulips
- The tulips need to be shown in different locations
- These tulips need to be yellow or red, so a tulip should have a color attribute
- The tulip should also be able to open and close based on user input, so it needs a open() and close() function
- A 'cloud' class is needed for the clouds
- The clouds should move
- A main class is needed to create and draw the clouds, tulips, and background

A possible design could be:



Class diagram (left) and use case diagram (right) of the blooming flower system



Sequence diagram of the blooming flowers system

For this simple example, the design includes a class diagram, interaction (sequence) diagram, and a use case diagram, as well as textual explanation. The textual explanation could also be provided verbally for this assignment.

Students can use all kinds of tools for this assignment. Possible tools are: draw.io, Paint, Photoshop, Visual Paradigm (used for the example, students can get it for free) but also pen and paper.

The design (including all diagrams) should be approved by a TA before students can continue to the next assignment.

Lecture 2.1: Introduction to assignment 2 and 3

The second and third assignments are introduced. For the second assignment, the importance of writing code that is understandable for others is stressed since this is important for the third assignment.

Lecture 2.2: Good coding habits and common mistakes

A live coding session will be held, in which a teacher will produce code step-by-step. During this session, the teacher will explain why they are doing what they are doing, which pitfalls are avoided, and why certain structural choices are made. The coding session should have a clear structure: there should be a clear goal that the teacher is working towards, so students know what to expect. The sketch itself does not need to be complicated, simple examples are easier to understand. It is up to the teacher to decide how exactly the coding session will be shaped. The session should be recorded so students can watch it back.

The session could cover an example design also used in lecture 1.2. The teacher will clearly explain how they would go from this design to code, and which choices are made along the way.

Lecture 2.3: Introduction to Git

To help students in their cooperation, an introduction to the simple functionality of Git is given. The focus is on making sure students know how to set up a repo, commit and push their code, and how a team member can pull these changes. For this purpose, (SNT) GitLab will be introduced. All students have free access to this, so they can use the information they learn during their whole student career. During the day this might not be as important, but the 6

information is useful for students who are going to work on their code in the evening, and of course for future projects. It is not the idea that GitLab will be used to hand in code, Canvas can be used for this.

For this lecture, it is important to distinguish between Git, GitLab, and GitHub.

Assignment 2

For the second assignment, students turn their own design into software. The code should be understandable for other students, without verbal explanation. After the software has been developed, the students need to hand in a revised version of their design as well.

The goal of this assignment is to let students come in contact with the process of writing code based on an existing design with certain requirements and the process of revising their own design. They might come across shortcomings within their design, determine that their original design is not feasible after all, or that some explanations might be missing. This will force them to think about their design and make new iterations of the code and the design.

Students are required to use Processing since this is the tool they are familiar with from previous modules and because it allows students to put their visual design and mental model in a visual context, which could help with the understanding of the program and programming concepts as a whole. If they have a clear preference for another programming language, they can use another language in consultation with the teaching staff.

Below an example piece of code based on the original example design made for assignment one can be seen.

```
Flower_blooming class
/*
Example code for the software design workshop of the graduation project
of Hannah Ottenschot on software design within Creative Technology.

May 2021

The code portrays moving clouds and three tulips, which bloom once the left mouse button is
pressed.
*/

color skyColor = color(0, 165, 255);
color grassColor = color(91, 194, 54);

Tulip yellowTulip;
Tulip redTulip1;
Tulip redTulip2;

Cloud cloud1;
Cloud cloud2;
Cloud cloud3;
```

```

boolean pressed;

void setup() {
    size(1080, 780);

    //Creating the tulips
    yellowTulip = new Tulip("yellow", width/2, height/2);
    redTulip1 = new Tulip ("red", (width/3)*2, height/3);
    redTulip2 = new Tulip ("red", width/5, (height/5)*2);

    //Creating the clouds
    cloud1 = new Cloud(200, 1);
    cloud2 = new Cloud(150, 3);
    cloud3 = new Cloud(100, 2);
}

void draw() {
    //Drawing the sky and grass
    background(skyColor);
    fill(grassColor);
    noStroke();
    rect(0, height/2, width, height);

    //Drawing the tulips
    displayFlowers();

    //Changing the flower on mouse click
    if (pressed) {
        displayFlowers();
    } else {
        closeFlowers();
    }

    //Drawing and moving the cloud
    displayClouds();
    driveClouds();
}

void mousePressed() {
    pressed = true;
    openFlowers();
}

void mouseReleased() {
    closeFlowers();
}

void displayFlowers() {
    yellowTulip.display();
}

```

```

    redTulip1.display();
    redTulip2.display();
}

//Created a void closeFlowers and openFlowers to address all flowers in one go
void closeFlowers() {
    yellowTulip.closeFlower();
    redTulip1.closeFlower();
    redTulip2.closeFlower();
}

//Changes the pictures of the flowers to an open state
void openFlowers() {
    yellowTulip.bloom();
    redTulip1.bloom();
    redTulip2.bloom();
}

//Display and move the clouds
void displayClouds() {
    cloud1.display();
    cloud2.display();
    cloud3.display();
}

void driveClouds() {
    cloud1.drive();
    cloud2.drive();
    cloud3.drive();
}

```

```

class Cloud {

    int xPos = 0;
    int yPos = 0;
    int size;
    int speed;

    PFigure cloud;

    Cloud(int resize, int xSpeed) {
        size = resize;
        speed = xSpeed;
    }

    void display() {
        //Loading in the cloud Figure and resizing it
        imageMode(CORNER);
        cloud = loadImage("Cloud.png");
        cloud.resize(0, size);
        image(cloud, xPos, yPos);
    }
}

```

```

}

void drive() {
    //Moving the clouds
    xPos += speed;
    if (xPos > width) {
        xPos = 0 - size * 2;
    }
}
}
}

```

```

class Tulip {
    color tulipColor;
    int xPos;
    int yPos;
    String colour;

    PImage tulip;

    String redClosed = "RedClosed.png";
    String yellowClosed = "YellowClosed.png";

    String redOpen = "RedOpen.png";
    String yellowOpen = "YellowOpen.png";

    Tulip(String colour, int xPos, int yPos) {
        this.colour = colour;
        this.xPos = xPos;
        this.yPos = yPos;

        if (colour == "red") {
            tulip = loadImage(redClosed);
        } else {
            tulip = loadImage(yellowClosed);
        }
    }

    void closeFlower() {
        if (colour == "red") {
            tulip = loadImage(redClosed);
        } else {
            tulip = loadImage(yellowClosed);
        }
    }

    void bloom() {
        if (colour == "red") {
            tulip = loadImage(redOpen);
        } else {

```



```

        tulip = loadImage(yellowOpen);
    }
}

void display() {
    //Drawing the stem
    imageMode(CENTER);
    stroke(0, 128, 0);
    strokeWeight(10);
    line(xPos, yPos, xPos, (yPos/3)*5);

    //Drawing the flower
    image(tulip, xPos, yPos);
}
}

```

Day 2

Assignment 3

For the third assignment, students will receive code from another group made for the second assignment. From the received code, the students need to again create a design. This design should also include a class, interaction, and use case diagram. Groups who received version A of the textual description will receive code from a group that made a program for version B. This will ensure that students do not have to evaluate code they have been looking at for quite some hours already, decreasing the chance that students will forcefully write feedback based on what they did themselves.

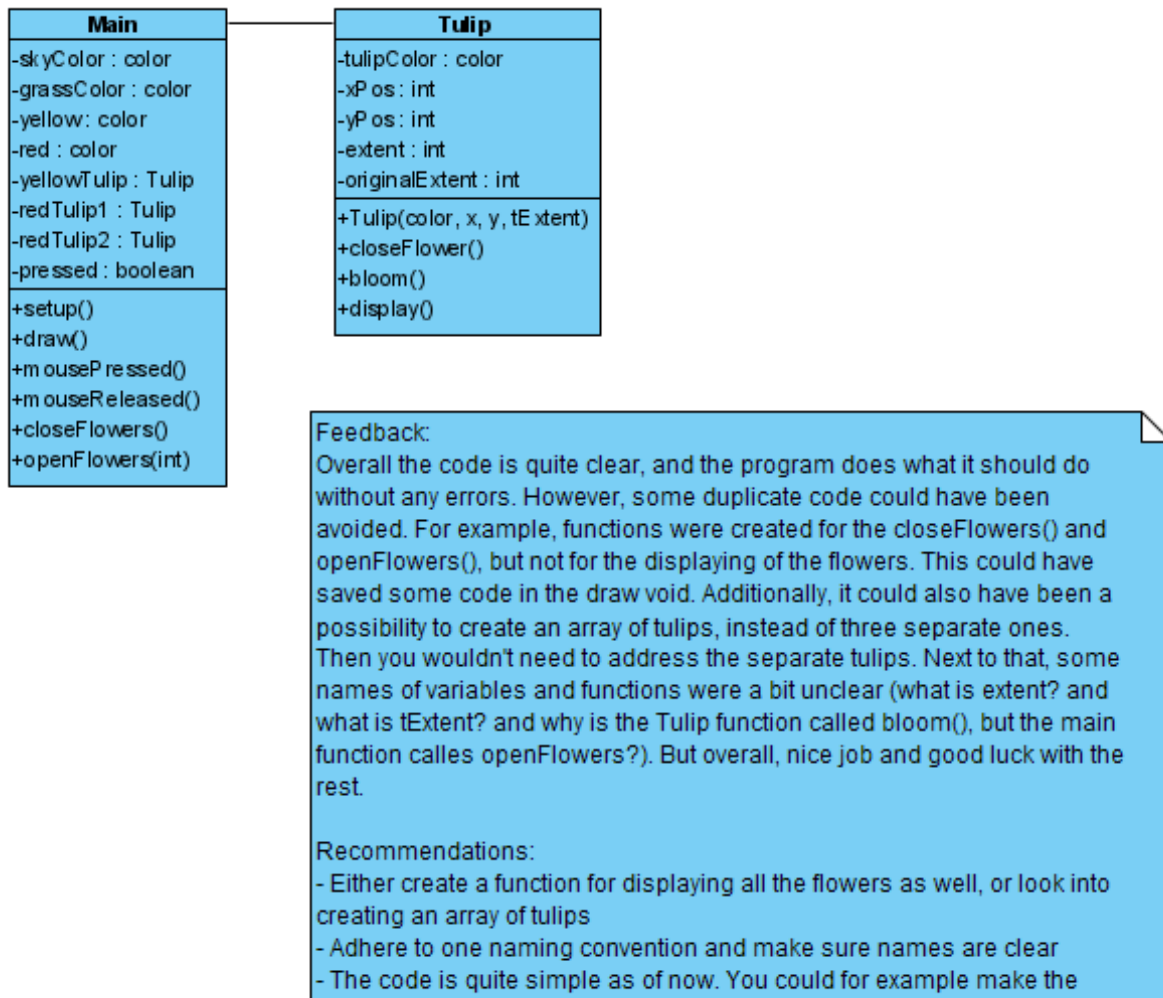
The teaching staff will make sure each student team will receive the code from another group. This makes sure that the students do not get the opportunity to explain their code verbally to the other team, as it should be self-explanatory from the documentation.

Additionally, it is required to provide feedback and a list of recommendations to the team who originally wrote the code. This can be about unclarities (documentation, naming conventions), the design of the code, the quality of code, if any pitfalls can be detected, etc. The same tools can be used that were used for the first assignment. The feedback will be delivered to the receiving team both textually as well as verbally, to make sure that it's clear what the providing students mean.

The goal of this assignment is to force students to critically assess a piece of code, and to help them in acquiring the necessary skills to judge their own code (or avoid pitfalls in the first place, of course) as well as that of others. Additionally, feedback- and recommendation-giving skills are trained. In general, this assignment will bring the students in closer contact with the real-life

practices of working with someone else's code as well as creating a design based on software that is understandable for others.

Below an example class diagram with feedback can be found. The example is based on the example code example of the second assignment. The interaction and use case diagrams have been left out since they are not that complex for this assignment.



Updated class diagram of the blooming flowers system

2. Schedule

Day 1	Activity	Notes
9:30 - 10:00	Lecture 1.1: Introduction of the topic software design	
10:00 - 10:15	Lecture 1.2: explanation of the first assignment	Examples should be given
10:15 - 12:30	Students work on the first assignments	Student groups of 2. TAs need to check the final results, provide feedback and approve of them
12:30 - 13:30	Break	
13:30 - 13:50	Lecture 2.1: Explanation of the second and third assignment	Stress the importance of understandable code
13:50 - 14:30	Lecture 2.2: good coding habits and common programming mistakes	Live coding session where a teacher talks about their thought process in detail
14:30 - 15:00	Lecture 2.3: Introduction to Git	Only the necessities: how do I work together on code efficiently If students already know how to do this, they can already start working on the second assignment
15:00 - 17:30	Students work on the second assignment	TAs are available to provide feedback. TAs do not need to approve the code
Day 2	Activity	Notes
9:00 - 11:30	Students can work on the second assignment	TAs are available to help students.
11:30 - 12:00	Teachers match teams and hand out code	The deadline for the assignment is 11:30. This gives time for teachers to match teams that hand in their code. Students should not explain their code verbally to the other team, since it should be understandable without. Having teachers hand out the code helps with this.

12:00 - 12:30	Students work on the third assignment	<p>TAs again need to check the final results and provide feedback.</p> <p>Students need to have an approved version of the first assignment, sent out their second assignment, and handed in their third assignment before they can go.</p>
12:30 - 13:30	Break	
13:30 - 17:15	Students continue working on the third assignment	
17:15 - 17:30	Closing	
Afterward	Activity	Notes
-	Giving marks on the third assignment	<p>If the original code is really bad, the students making a design based on it should not be punished for this. The students should be graded based on whether they make logical assumptions based on the original code, and whether their recommendations for the other group are logical and constructive</p> <p>Once the grades are known, the students receive them with the graded recommendations.</p>

3. Role of Introduction to Software Design within CreaTe curriculum

3.1 Learning goals

The intended learning goals of the workshop are diverse. First and foremost, the workshop is meant to introduce students to the topic of software design: the process of creating an initial design, writing code based on the initial design and explaining what code does in understandable visuals and language. Additionally, the focus during this process is on real-life practices: what do real software engineers and designers do and how do they communicate within their work? This will not only help students during their Creative Technology career but also in their future working field.

The first assignment focuses on the creation of a design based on external requirements. This puts students in a position where the general outline and idea are already given, and the only thing that is left is to structure the information clearly and concisely. While this does take away from the creativity a student has, it is more related to the real-life practices of a programmer. Additionally, by initially creating a design, students get the chance to create a mental model of the software project they will have to tackle. This will help them in understanding the programming concepts that are needed to get to the final product and give them a visual presentation to fall back on when they get stuck. These concepts are introduced by the first two lectures (1.1 and 1.2). TAs need to approve of the design before students can continue to make sure that they get the necessary feedback and can iterate when needed., as well as that students can be confident that their design is correct.

During the second assignment, students need to create code based on their own design. Here they can utilize the mental model they made during the previous assignment, and show their abilities in creating a piece of software. To help them efficiently and effectively write code, a lecture (2.1) will be given on good coding habits and common mistakes. The concepts introduced will hopefully be useful to the students during all of their programming careers, and can help them avoid making mistakes. Additionally, Git will be introduced as a tool to work together on code, as it is an almost universally used tool and can be used during all CreaTe modules to come. Simple Git concepts will be introduced during lecture 2.3, as well as how to work with GitLab. For this assignment, the necessity of writing code that is understandable for others, people outside the project group you work with, is stressed to make students understand the importance.

For the third assignment, students' code is given to other teams. They do not get the chance to explain their code verbally, as the documentation and naming conventions should be understandable for the other team without explanation. From the received code, students need to create a design that explains the workings of the code. Similar to the first assignment, this will help students with the process of creating a design. Contrary, the design will be based on

code instead of text. This will help students with their ability to effectively explain what a piece of code is about and what the intended outcomes are. Additionally, students have to provide feedback and a list of recommendations for the other team. This can be about the general structure of the code, whether the functions used are efficient and logical if the syntax is correct, naming conventions, etc. This will help students critically analyze code and find pieces of code that are inefficient, which in turn can help them get better at programming.

3.2 Connection to module 3

As of now, there are no programming courses within module 3, making it hard to integrate the workshop with an existing course. However, students have indicated that the gap between modules 2 and 4 when it comes to programming is quite big. Many students tend to forget programming concepts since they do not need to focus on the topic. By introducing software design within the 3rd module together with a programming assignment, students are forced to focus on programming. This can help them prepare for module 4, which is often seen as a difficult module.

3.3 Connection to other modules

The concept described in the Learning Goals chapter can be applied during almost every CreaTe module. By introducing the topic (relatively) early, students can apply their skills for the rest of their CreaTe career, and within their work.

Appendix 5: Evaluation workshop content

5.1: What is software design?



Figure 13: Slide 1 of the “What is software design?” presentation for the evaluation workshop

What is software design?

- ‘Design’ choices
- Abstractions on code level
- System behaviour



Figure 14: Slide 2 of the “What is software design?” presentation for the evaluation workshop

Why is software design important?

- Influence on outcome
- Performance, Maintainability
- Quality of code, time needed to fix bugs
- Understandability
- Asking targeted questions



Figure 15: Slide 3 of the “What is software design?” presentation for the evaluation workshop

Real-life examples

- Unified Modelling Language (UML)
- Informal examples
- Diagrams:
 - Class
 - Interaction
 - Use case



Figure 16: Slide 4 of the “What is software design?” presentation for the evaluation workshop

Class diagram

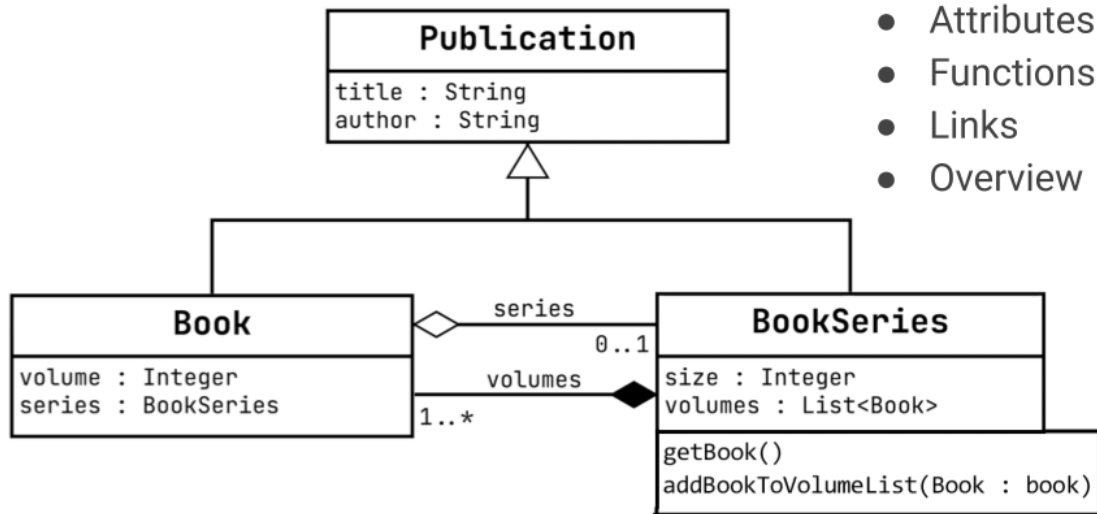


Figure 17: Slide 5 of the “What is software design?” presentation for the evaluation workshop

Interaction diagram

- Dynamic behaviour
- Different parts of the system
- Actors
- Interaction
- Accessible to non-coders
- Not code language dependent

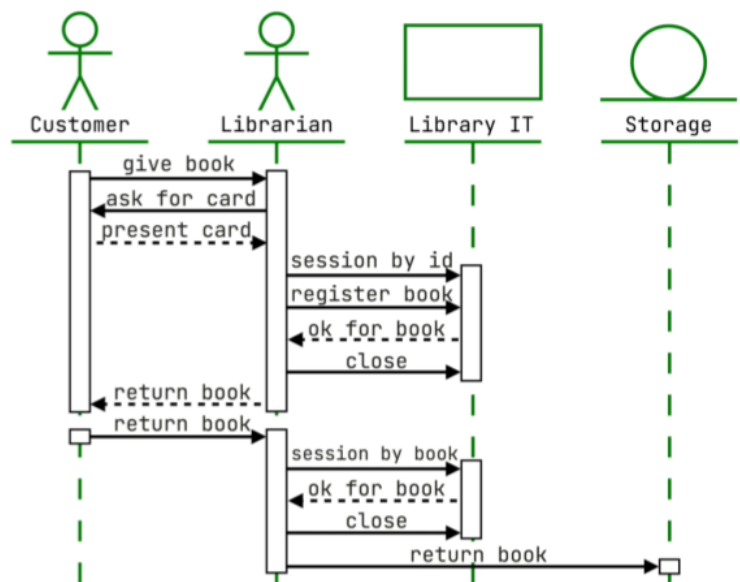


Figure 18: Slide 6 of the “What is software design?” presentation for the evaluation workshop

Use case diagram

- System
- Actors
- Use cases
- Links
- Checking requirements
- Easy to understand

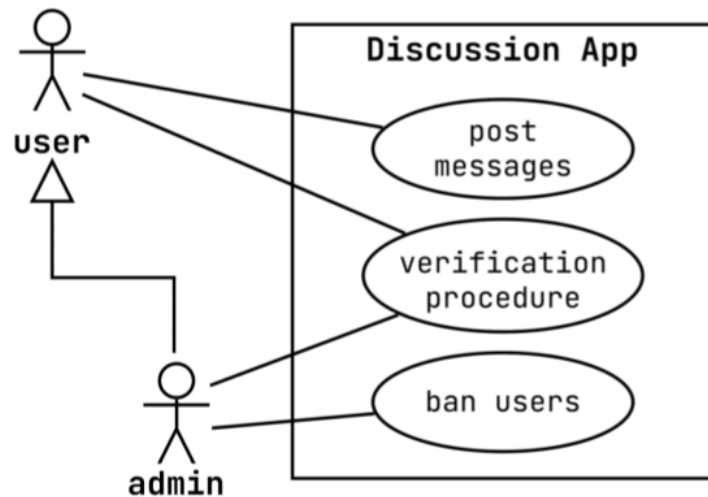


Figure 19: Slide 7 of the “What is software design?” presentation for the evaluation workshop

5.2: Assignment 1

ASSIGNMENT 1

Create a design from a textual description

Software Design workshop

Hannah Ottenschot

Figure 20: Slide 1 of the “Assignment 1” presentation for the evaluation workshop

From text to design

- You will receive a situation and textual description of the code
- The following should be included in your design:
 - Class diagram
 - Interaction diagram
 - Use case diagram
- The design can be made using an online tool (draw.io), application (Paint, Photoshop, Visual Paradigm) or using pen and paper



Figure 21: Slide 2 of the “Assignment 1” presentation for the evaluation workshop

Example - description

Sketch: Croaking frogs

This sketch should contain a small game with green frogs in a pond. The user controls a fly: by using the arrow keys, the user can move the fly across the screen. If the fly comes too close to a frog, the frog will shoot its tongue towards the fly. If the fly does not move in time, it will be caught by the frog, and the user will lose the game. Above the screen, the amount of time the user has survived is shown. If the player is game over, the total amount of survived time will be shown in the middle of the screen and the counter will stop increasing. The user is given the option to start the game again.

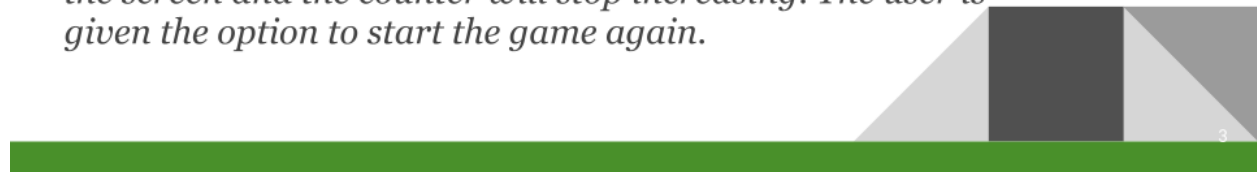


Figure 22: Slide 3 of the “Assignment 1” presentation for the evaluation workshop

Example - elicited requirements

- A 'frog' class is needed for the frogs
- The frogs need to be green and shown on different locations
- Frogs should be able to detect a bypassing fly
- Frogs should be able to catch flies
- A 'fly' class is needed for the fly
- The fly should be able to fly based on user input
- A timer is needed to show the amount of survived time
- The timer should increase as long as the player is alive
- A main class is needed to create and draw the frogs, flies and background and deal with interaction
- The user should be able to see their score and replay the game

Figure 23: Slide 4 of the "Assignment 1" presentation for the evaluation workshop

Example - class diagram

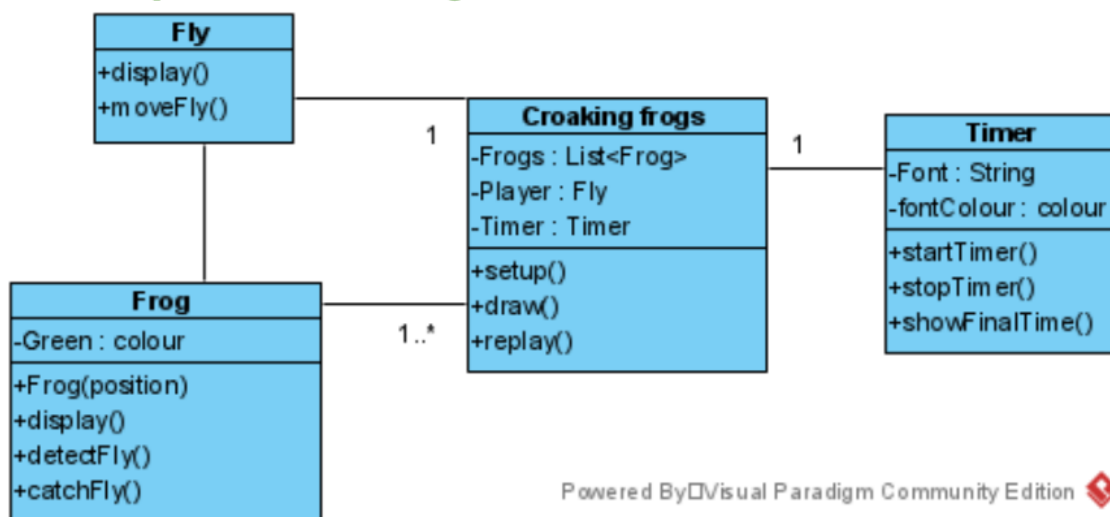


Figure 24: Slide 5 of the "Assignment 1" presentation for the evaluation workshop

Example - interaction diagram

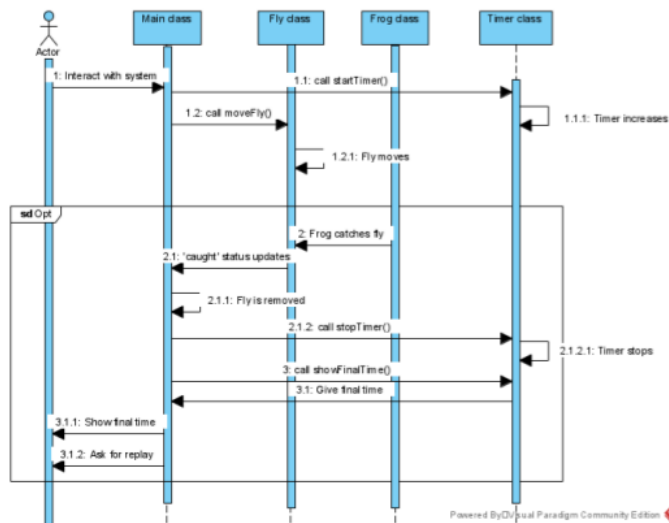


Figure 25: Slide 6 of the “Assignment 1” presentation for the evaluation workshop

Example - use case diagram

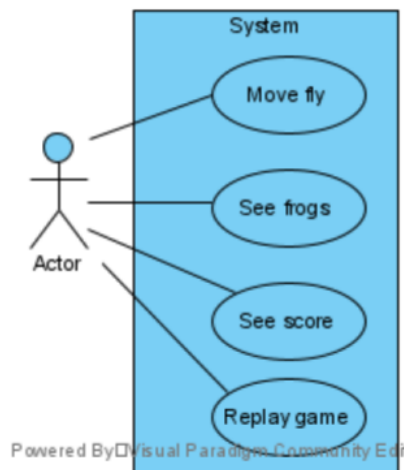


Figure 26: Slide 7 of the “Assignment 1” presentation for the evaluation workshop

Assignment

Sketch: Buzzing bees

This sketch should portray bees in a flower field. The background of the sketch should show the sky with moving clouds, a green grass field and a nice Spring sun. On the grass field, multiple flowers can be seen on random locations. Bees should fly over the screen in random directions and with random speeds. If the user clicks on a flower, nectar appears on this flower. Bees flock towards this flower, after which the nectar disappears and the bees return to flying in random directions.

8

Figure 27: Slide 8 of the “Assignment 1” presentation for the evaluation workshop

5.3: Explanation of original workshop

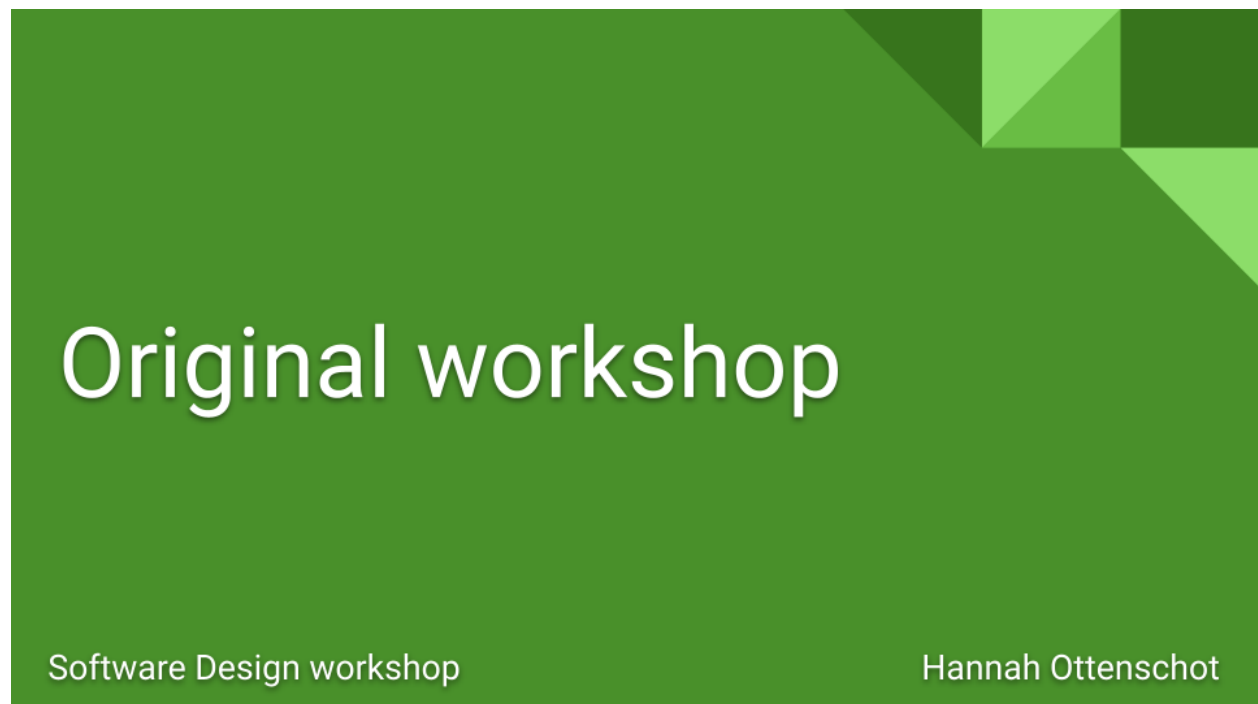


Figure 28: Slide 1 of the “Explanation of original workshop” presentation for the evaluation workshop

Original workshop

- 1.5 days → more time and possible complexity
- Groups of 2
- Assignments
 - 1: Make a design from textual description
 - **2: Write code based on your own design**
 - 3: Make a design based on other students' code
- Lectures
 - Introduction to software design
 - Good coding habits **and common mistakes (live coding session)**
 - **Introduction to Git**

Figure 29: Slide 2 of the “Explanation of original workshop” presentation for the evaluation workshop

Schedule of today

Explanation of original workshop	5
Intro to software design	10
Intro to assignment 1	5
Work on assignment 1	30
Intro to assignment 2	5
Lecture on good coding habits	10
Work on assignment 2	30
Discuss feedback for second assignment	10
Closing & evaluation	15
	120 min
	2 hours

Figure 30: Slide 3 of the “Explanation of original workshop” presentation for the evaluation workshop

5.4: Good coding habits

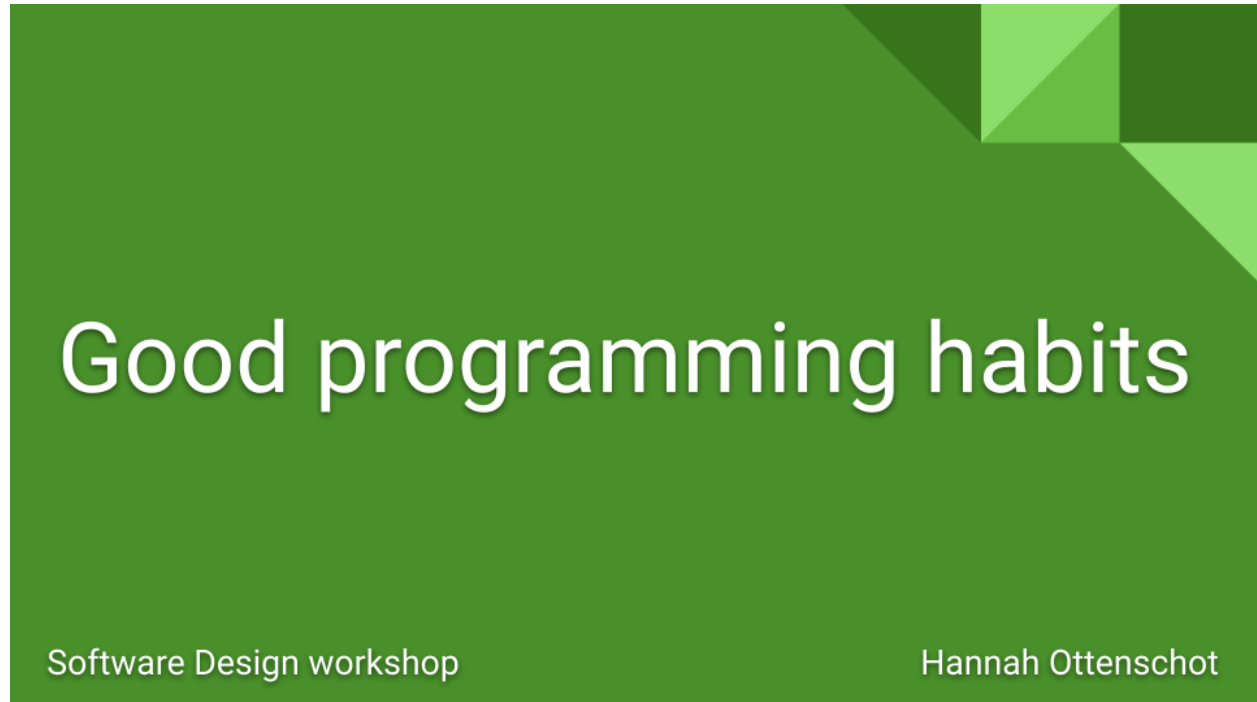


Figure 31: Slide 1 of the “Good coding habits” presentation for the evaluation workshop

Why learn good programming habits?

- Become a better programmer
- More clarity for yourself
- More clarity for others
- Communication



Figure 32: Slide 2 of the “Good coding habits” presentation for the evaluation workshop

Effects of good programming habits?

- Quality code
 - Clear to understand
 - Pleasant to work with
 - Easy to change
 - No hardcoding!



Figure 33: Slide 3 of the “Good coding habits” presentation for the evaluation workshop

General tips

- Keep your code clean
 - Comment where necessary
 - Note: comments do not compensate for bad code!
 - Remove dead code
- Use classes for separate concerns (encapsulation)
 - Keep classes small
 - Isolated functionality is good
 - Nothing should do everything

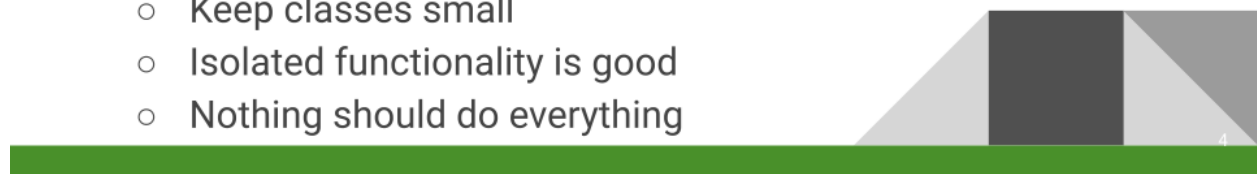


Figure 34: Slide 4 of the “Good coding habits” presentation for the evaluation workshop

General tips

- Give meaningful names and headers to classes, variables and functions that are
 - Understandable
 - Revealing the intentions
- Write code once
 - Watch out with copying code



Figure 35: Slide 5 of the “Good coding habits” presentation for the evaluation workshop

General tips

- Write short functions
 - 10 - 20 lines
 - In practice: 95% contains 1 - 15 lines
- Write simple functions
 - Do not branch too much
 - Do not use if-loops in if-loops in if-loops
 - Keeps functions easy to analyse, test and debug



Figure 36: Slide 6 of the “Good coding habits” presentation for the evaluation workshop

5.5: Assignment 2



Figure 37: Slide 1 of the “Assignment 2” presentation for the evaluation workshop

From code to design

- You will receive Processing code from me
- Again make a design, including three different diagrams: class, interaction and use case
- Additionally, provide feedback and a list of recommendations for the code
- We will discuss the feedback after everyone is done
- Afterwards there is time for general feedback on this workshop

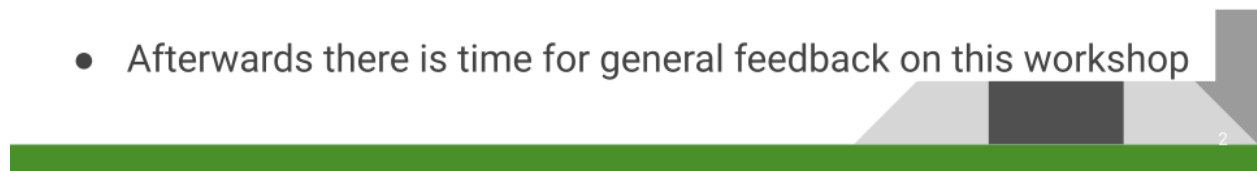


Figure 38: Slide 2 of the “Assignment 2” presentation for the evaluation workshop

5.5.1: Code

The following sketch was made based on this description:

Sketch: Flowers in bloom

This sketch should portray flowers blooming. Three closed tulips are shown in different locations: one red, and two yellow ones. When a child interacts with the system, the flowers should open. Once the child stops interacting, the flowers should close again. The background of the sketch should show the sky with moving clouds and a green grass field.

There are three classes, Flower_Blooming, Cloud, and Tulip. Your assignment is to evaluate this code and give feedback and a list of recommendations.

Picture of the output:

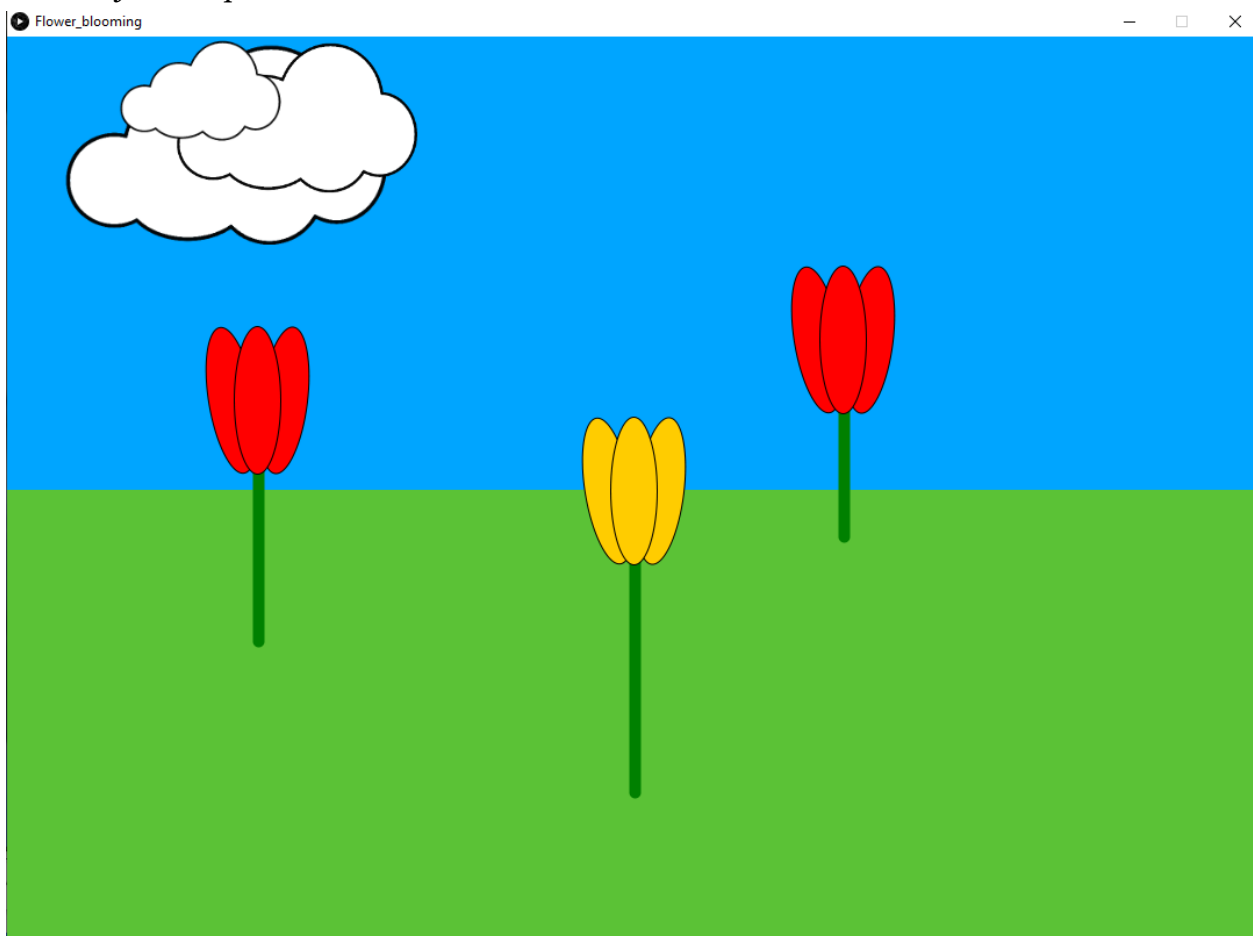


Figure 39: Output of the code given to evaluation workshop participants

```
Flower_blooming class
/*
Example code for the software design workshop of the graduation project
of Hannah Ottenschot on software design within Creative Technology.
```

May 2021

The code portrays moving clouds and three tulips, which bloom once the left mouse button is pressed.

```
*/

color skyColor = color(0, 165, 255);
color grassColor = color(91, 194, 54);

Tulip yellowTulip;
Tulip redTulip1;
Tulip redTulip2;

Cloud cloud1;
Cloud cloud2;
Cloud cloud3;

boolean pressed;

void setup() {
    size(1080, 780);

    //Creating the tulips
    yellowTulip = new Tulip("yellow", width/2, height/2);
    redTulip1 = new Tulip ("red", (width/3)*2, height/3);
    redTulip2 = new Tulip ("red", width/5, (height/5)*2);

    //Creating the clouds
    cloud1 = new Cloud(200, 1);
    cloud2 = new Cloud(150, 3);
    cloud3 = new Cloud(100, 2);
}

void draw() {
    //Drawing the sky and grass
    background(skyColor);
    fill(grassColor);
    noStroke();
    rect(0, height/2, width, height);

    //Drawing the tulips
    displayFlowers();

    //Changing the flower on mouse click
    if (pressed) {
        displayFlowers();
    } else {
        closeFlowers();
    }

    //Drawing and moving the cloud
```

```

    displayClouds();
    driveClouds();
}

void mousePressed() {
    pressed = true;
    openFlowers();
}

void mouseReleased() {
    closeFlowers();
}

void displayFlowers() {
    yellowTulip.display();
    redTulip1.display();
    redTulip2.display();
}

//Created a void closeFlowers and openFlowers to address all flowers in one go
void closeFlowers() {
    yellowTulip.closeFlower();
    redTulip1.closeFlower();
    redTulip2.closeFlower();
}

//Changes the pictures of the flowers to an open state
void openFlowers() {
    yellowTulip.bloom();
    redTulip1.bloom();
    redTulip2.bloom();
}

//Display and move the clouds
void displayClouds() {
    cloud1.display();
    cloud2.display();
    cloud3.display();
}

void driveClouds() {
    cloud1.drive();
    cloud2.drive();
    cloud3.drive();
}

```

```

class Cloud {

    int xPos = 0;
    int yPos = 0;
    int size;
    int speed;

    PFigure cloud;

    Cloud(int resize, int xSpeed) {
        size = resize;
        speed = xSpeed;
    }

    void display() {
        //Loading in the cloud Figure and resizing it
        imageMode(CORNER);
        cloud = loadImage("Cloud.png");
        cloud.resize(0, size);
        image(cloud, xPos, yPos);
    }

    void drive() {
        //Moving the clouds
        xPos += speed;
        if (xPos > width) {
            xPos = 0 - size * 2;
        }
    }
}

```

```

class Tulip {
    color tulipColor;
    int xPos;
    int yPos;
    String colour;

    PFigure tulip;

    String redClosed = "RedClosed.png";
    String yellowClosed = "YellowClosed.png";

    String redOpen = "RedOpen.png";
    String yellowOpen = "YellowOpen.png";

    Tulip(String colour, int xPos, int yPos) {

```

```

    this.colour = colour;
    this.xPos = xPos;
    this.yPos = yPos;

    if (colour == "red") {
        tulip = loadImage(redClosed);
    } else {
        tulip = loadImage(yellowClosed);
    }
}

void closeFlower() {
    if (colour == "red") {
        tulip = loadImage(redClosed);
    } else {
        tulip = loadImage(yellowClosed);
    }
}

void bloom() {
    if (colour == "red") {
        tulip = loadImage(redOpen);
    } else {
        tulip = loadImage(yellowOpen);
    }
}

void display() {
    //Drawing the stem
    imageMode(CENTER);
    stroke(0, 128, 0);
    strokeWeight(10);
    line(xPos, yPos, xPos, (yPos/3)*5);

    //Drawing the flower
    image(tulip, xPos, yPos);
}
}

```


Appendix 6: Evaluation workshop results

1: Pilot session

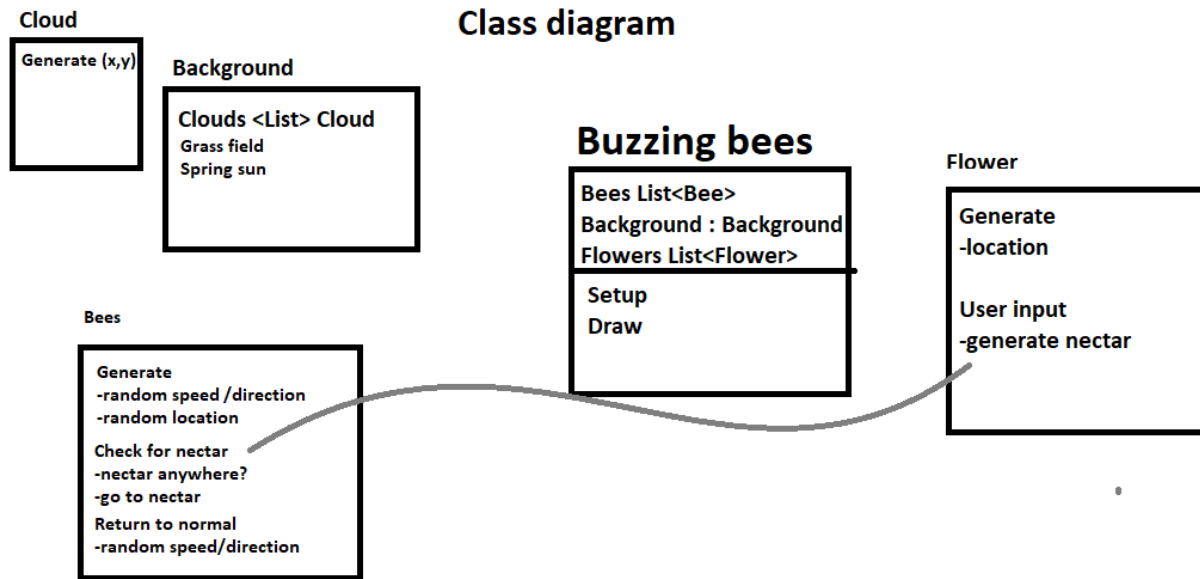


Figure 40: Class diagram made during the evaluation workshop pilot session

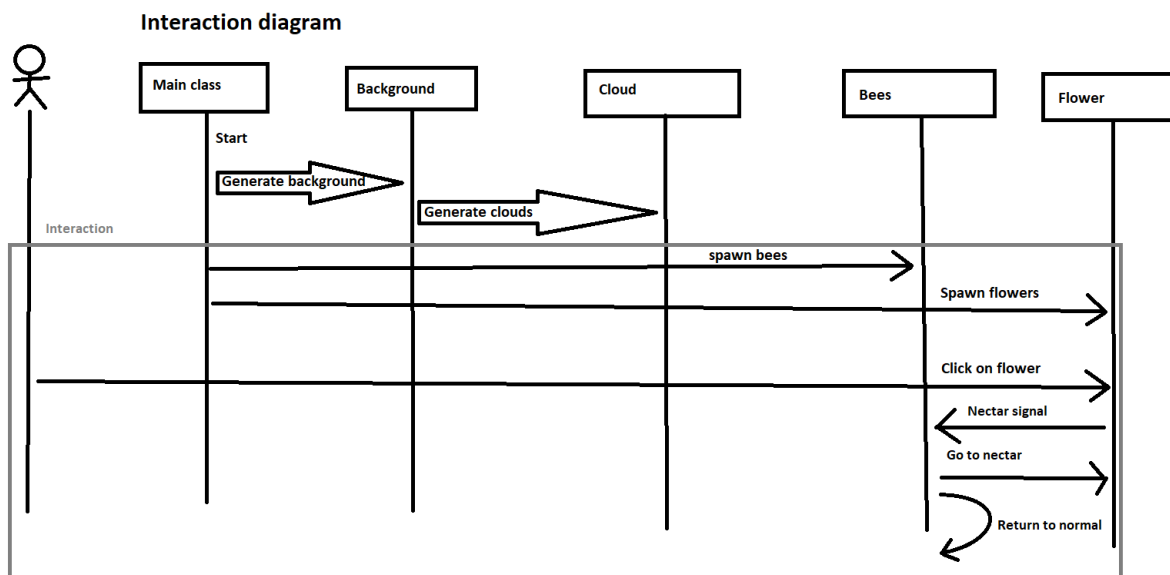


Figure 41: Interaction diagram made during the evaluation workshop pilot session

Use case diagram

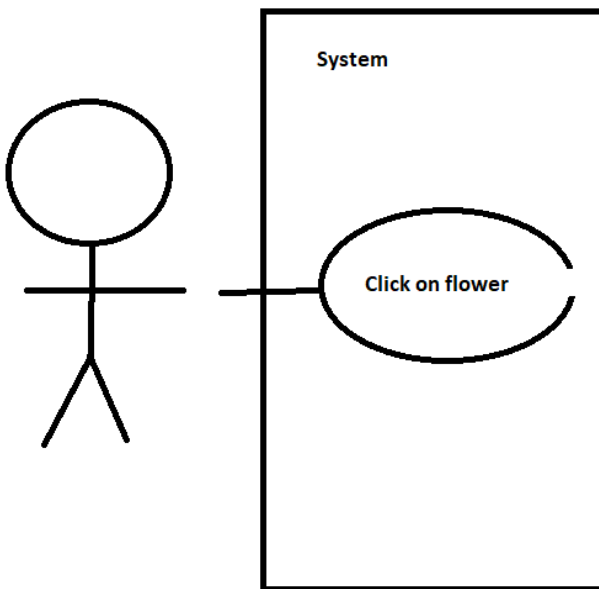


Figure 42: Use case diagram made during the evaluation workshop pilot session

2: Workshop 1

2.1: Workshop 1 - diagrams

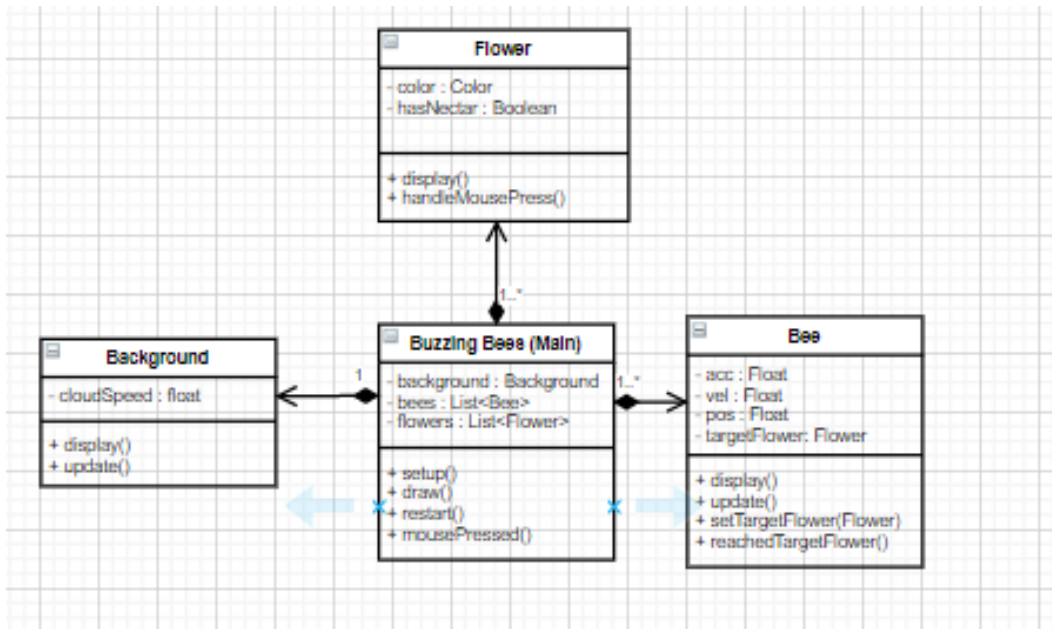


Figure 43: Class diagram made by participant 1 during evaluation workshop 1

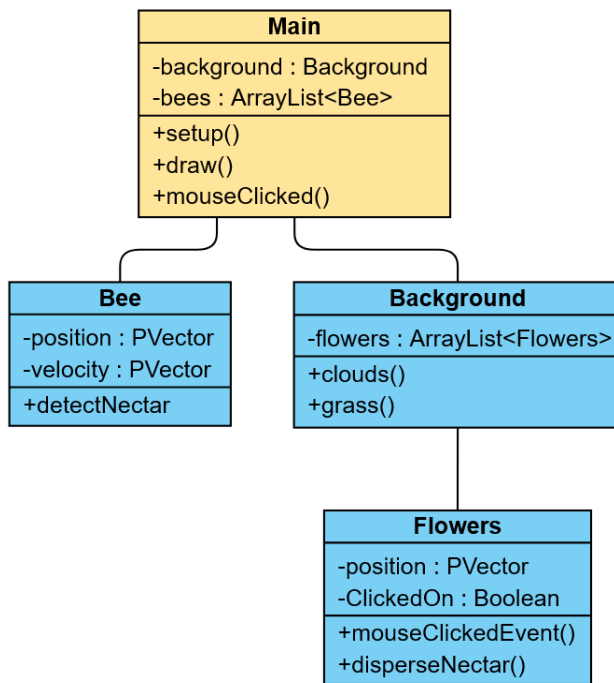


Figure 44: Class diagram made by participant 2 during evaluation workshop 1

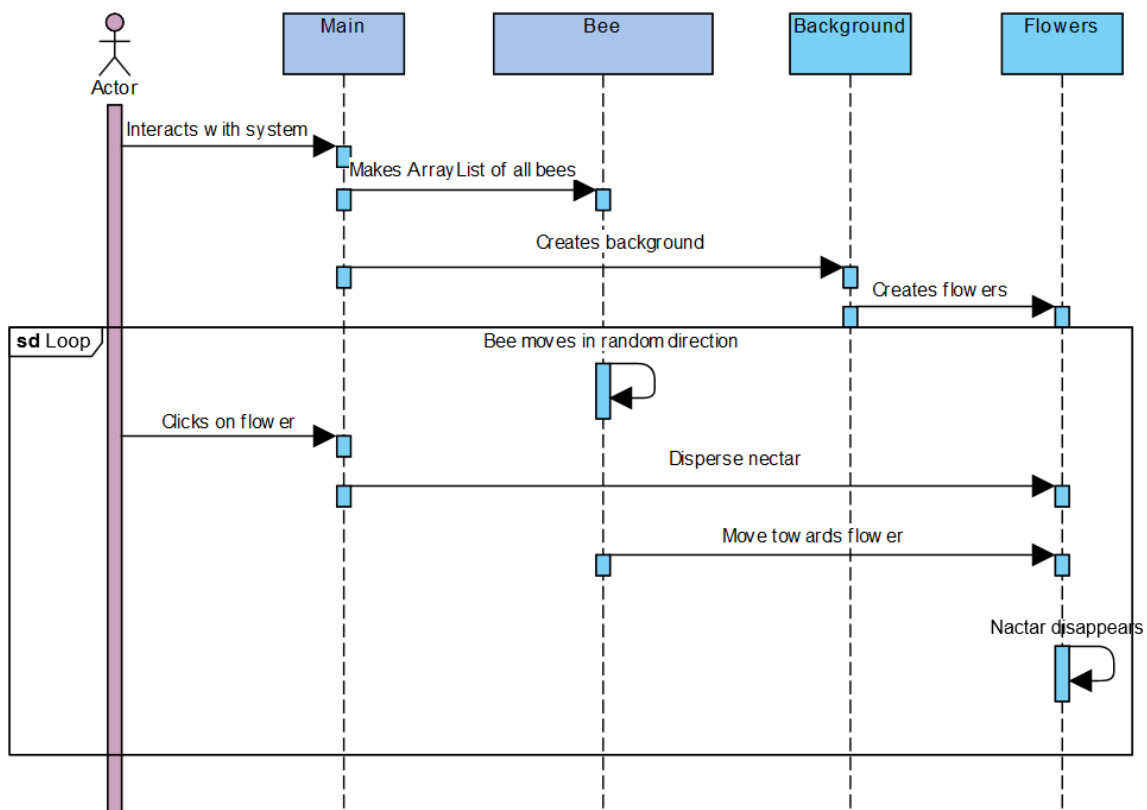


Figure 45: Interaction diagram made by participant 2 during evaluation workshop 1

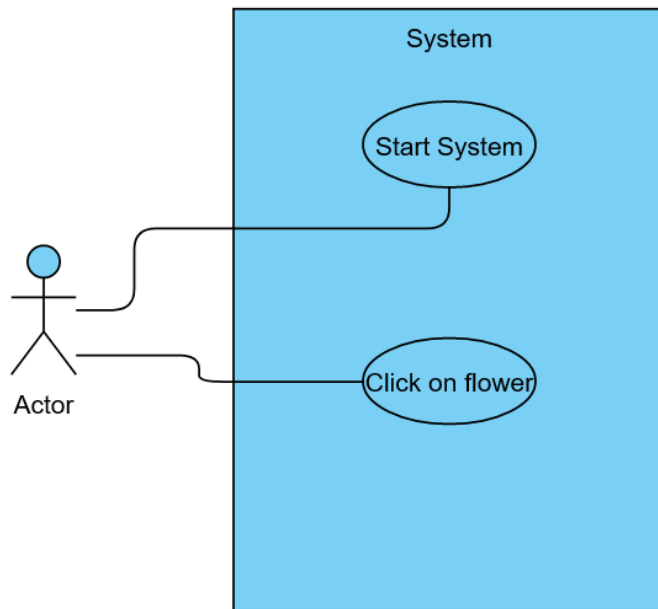


Figure 46: Use case diagram made by participant 2 during evaluation workshop 1

2.2: Workshop 1 - feedback

ARRAYS PLEASE
 A lot of duplicate code
 Main class has methods that call methods of the other classes
 Using strings to determine which colour Figure should be displayed?
 GLOBAL VARIABLES WTF
 Magic numbers~
 Main class does all kinds of things it shouldn't be doing.
 WEIRD (incorrect?) comments

Feedback given by participant 1 during evaluation workshop 1

- Make arrays of tulips of cloud
- displayFlowers(), closeFlowers() and openFlowers() methods in the main file and is a duplicate method of closeFlower() and bloom()
- Comments don't make sense a lot of the time
- Parse objects through methods
- Don't need to use PImages. Ellipses with a fill and rotate would be easier
- xPos = 0-yPos
- String Colour
- Make boolean method of 'Red' and 'Yellow'

Feedback given by participant 2 during evaluation workshop 1

3: Workshop 2

3.1: Workshop 2 - diagrams

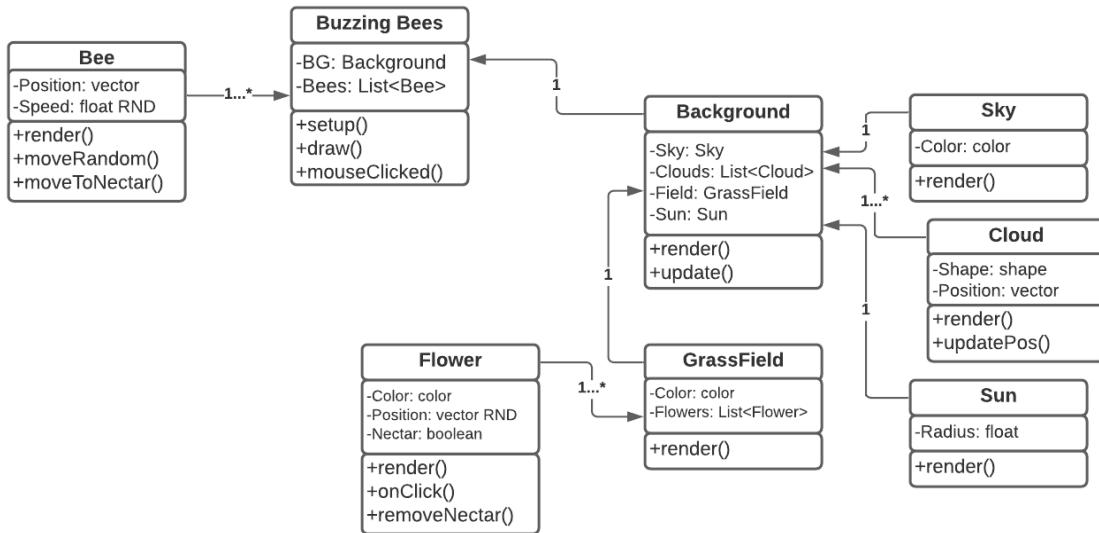


Figure 47: Class diagram made by participant 1 during evaluation workshop 2

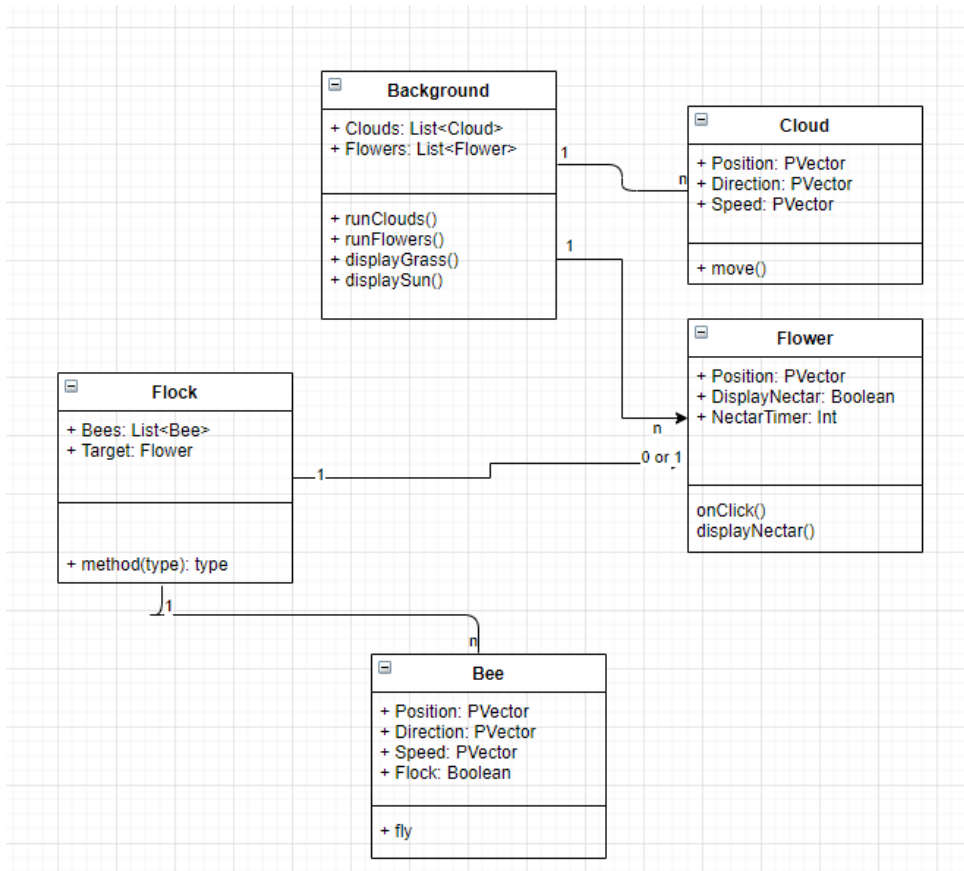


Figure 48: Class diagram made by participant 2 during evaluation workshop 2

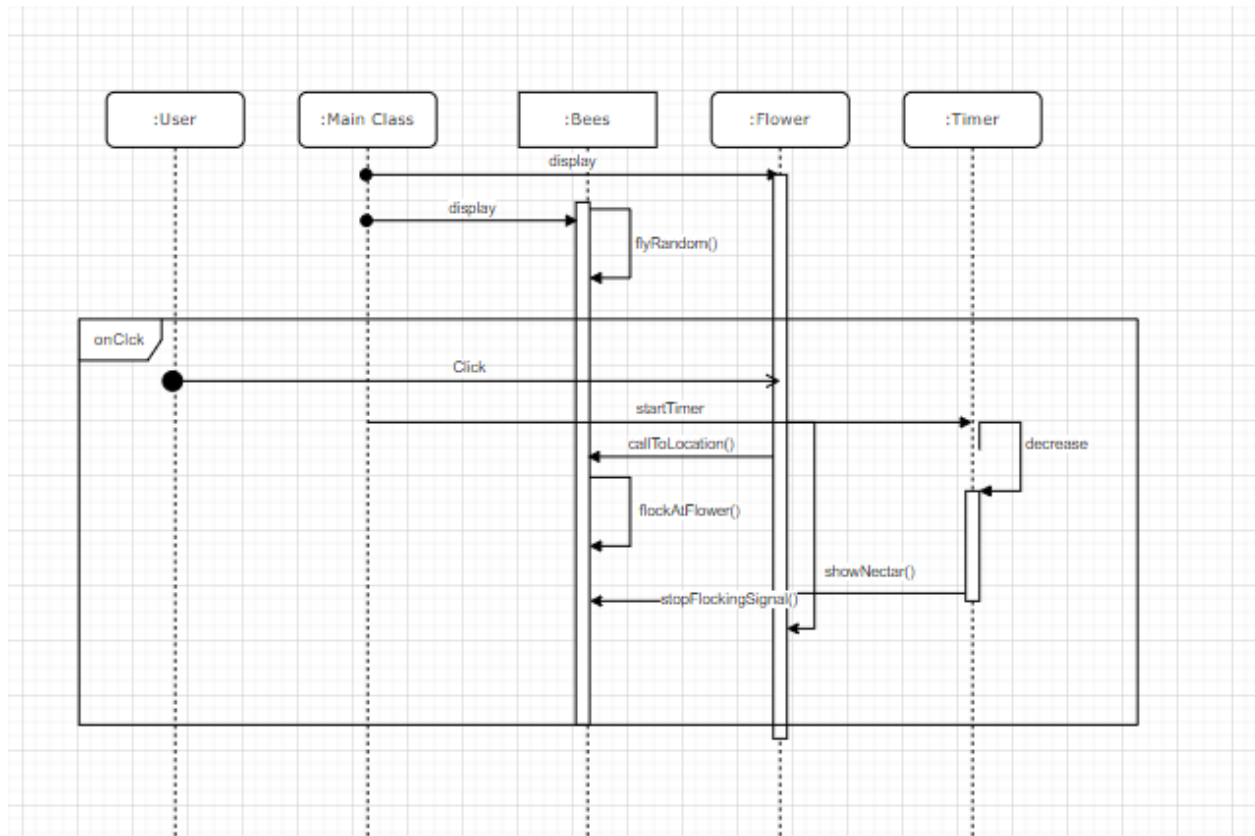


Figure 49: Interaction diagram made by participant 2 during evaluation workshop 2

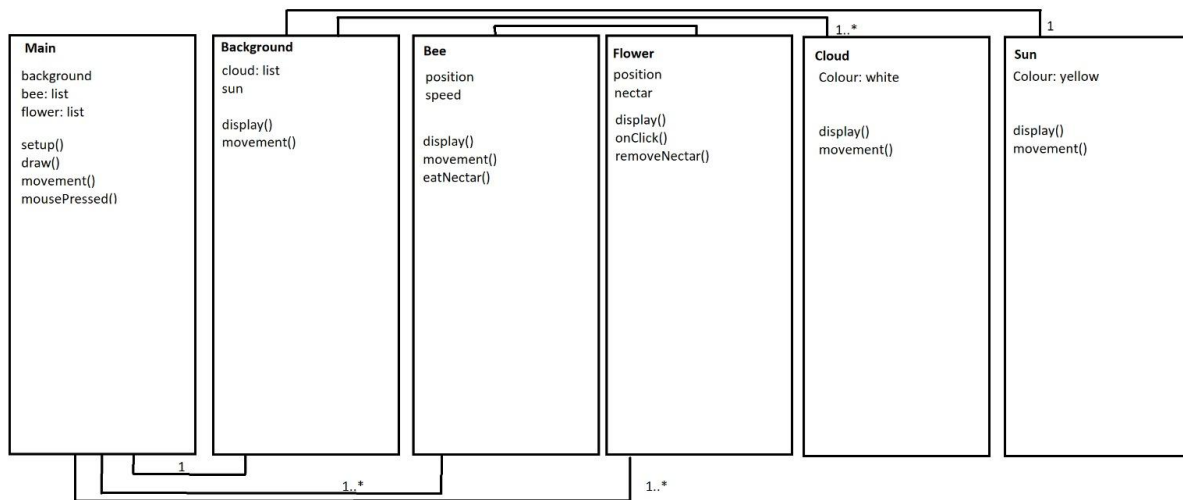


Figure 50: Class diagram made by participant 3 during evaluation workshop 2



Figure 51: Interaction diagram made by participant 3 during evaluation workshop 2

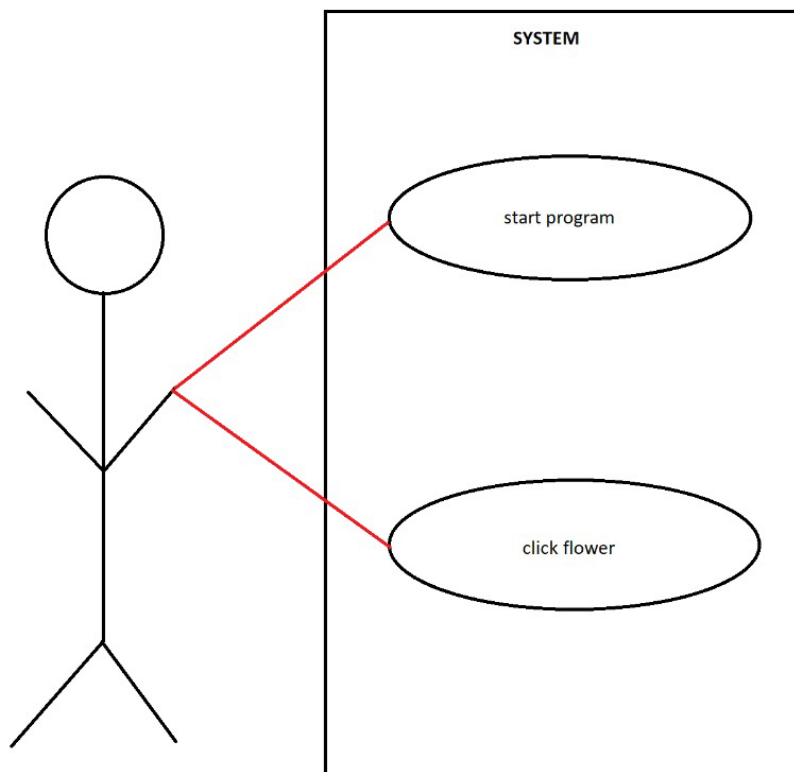


Figure 52: Use case diagram made by participant 3 during evaluation workshop 2

3.2: Workshop 2 - feedback

Use lists for multiple objects of the same class. (tulip1..., cloud1...)

Use handler classes to contain and manage the lists.

-> class Tulips with ArrayList<Tulip> list, and add(color, x, y)

-> class Clouds with ArrayList<Cloud> list, and add(x, y)

In my opinion, use classes for sky and field, to make the draw more easily readable.

Keep event handling out of the main loop, instead just update a state of the tulip objects, and render depending on that state in the Tulip render method.

Why are there two displayFlowers() calls in draw()?? Second should be openFlowers()?

Dont constantly reloadImages on every change. Preload them in setup and simply switch to drawing the right one using a state.

Use arrays for the different coloured and oppen/closed flower images and access them using integer constants/variables.

```
```java
```

```
final int FLOWER_RED = 0
```

```
final int FLOWER_YELLOW = 1
```

```
final int FLOWER_COLOR_VARIATIONS = 2
```

```
// States
```

```
// FLOWER_STATE_CLOSED = 0
```

```
// FLOWER_STATE_OPEN = 1
```

```
// this part would ideally be done in the constructor of the Tulips class (handler for all tulips objects,
// and then passed through the render method
```

```
final PImage[][] imgList = new PImage[2]
```

```
for (int i = 0; i < FLOWER_COLOR_VARIATIONS; i++)
```

```
 for (int j : [0, 1])
```

```
 imgList[i][j] = loadImage(i+'_'+j+'.png') // assuming the pngs would have different naming
```

```
// the values can also be mapped to 'Red'/'Yellow' and 'Open'/'Closed' instead
```

```
```
```

The program could use deltaTime to avoid different cloud drifting speeds depending on framerates.

Personally not a big fan of mixing American and British english (color and colour), especially since color is a method of Processing and colour isnt. So using colour as a variable can be confusing.

Feedback given by participant 1 during evaluation workshop 2


```
color skyColor = color(0, 165, 255);
color grassColor = color(91, 194, 54);
```

```
Tulip yellowTulip;
Tulip redTulip1;
Tulip redTulip2;
```

```
Cloud cloud1;
Cloud cloud2;
Cloud cloud3;
```

```
boolean pressed;
```

```
void setup() {
  size(1080, 780);

  //Creating the tulips
  yellowTulip = new Tulip("yellow", width/2, height/2);
  redTulip1 = new Tulip ("red", (width/3)*2, height/3);
  redTulip2 = new Tulip ("red", width/5, (height/5)*2);

  //Creating the clouds
  cloud1 = new Cloud(200, 1);
  cloud2 = new Cloud(150, 3);
  cloud3 = new Cloud(100, 2);
}
```

```
void draw() {
  //Drawing the sky and grass
  background(skyColor);
  fill(grassColor);
  noStroke();
  rect(0, height/2, width, height);

  //Drawing the tulips
  displayFlowers();
}
```

```
//Changing the flower on mouse click
if (pressed) {
  displayFlowers();
} else {
  closeFlowers();
}
```

```
//Drawing and moving the cloud
displayClouds();
driveClouds();
```

Highlight 26/05/2021 16:51:45
Margot Options

Global variables, put in appropriate classes instead

Highlight 26/05/2021 16:52:20
Margot Options

Why code colours as type String and not as type colour?

Highlight 26/05/2021 16:52:51
Margot Options

Make an array of objects instead

Highlight 26/05/2021 17:11:49
Margot Options

Just toggle the 'open' status in the Flower class from mousePressed();

```

}

void mousePressed() {
    pressed = true;
    openFlowers();
}

void mouseReleased() {
    closeFlowers();
}

void displayFlowers() {
    yellowTulip.display();
    redTulip1.display();
    redTulip2.display();
}

//Created a void closeFlowers and openFlowers to address all flowers in one go
void closeFlowers() {
    yellowTulip.closeFlower();
    redTulip1.closeFlower();
    redTulip2.closeFlower();
}

//Changes the pictures of the flowers to an open state
void openFlowers() {
    yellowTulip.bloom();
    redTulip1.bloom();
}

```

Highlight 26/05/2021 17:11:24

Margot

Options

Just toggle the 'open' status in the Flower class, like;

```

for (Flower f: flowers) {
    f.opened != f.opened
}

```

Highlight 26/05/2021 16:54:39

Margot

Options

For everything; should be functions to run over objects of an array

```

class Cloud {

    int xPos = 0;
    int yPos = 0;
    int size;
    int speed;

    PImage cloud;

    Cloud(int resize, int xSpeed) {
        size = resize;
        speed = xSpeed;
    }

    void display() {
        //Loading in the cloud image and resizing it
        imageMode(CORNER);
        cloud = loadImage("Cloud.png");
        cloud.resize(0, size);
        image(cloud, xPos, yPos);
    }

    void drive() {
        //Moving the clouds
        xPos += speed;
        if (xPos > width) {
            xPos = 0 - size * 2;
        }
    }
}

```

Highlight 26/05/2021 17:08:52 ✕
 Margot Options ▾
 loading image every frame....
 please use the loadImage() function
 only once (in the constructor), and then
 only image() on display()

```

class Tulip {
    color tulipColor;
    int xPos;
    int yPos;
    String colour;

    PImage tulip;

    String redClosed = "RedClosed.png";
    String yellowClosed = "YellowClosed.png";

    String redOpen = "RedOpen.png";
    String yellowOpen = "YellowOpen.png";

    Tulip(String colour, int xPos, int yPos) {
        this.colour = colour;
        this.xPos = xPos;
    }
}

```

Highlight 26/05/2021 16:55:42 ✕
 Margot Options ▾
 why a String?

Feedback given by participant 2 during evaluation workshop 2

- Visual doesn't match sketch (1 red & 2 yellow vs 2 red & 1 yellow)
- Specify interaction: click, drag, ...
- should all flowers open or only the one that is interacted with?

- skyColor and grassColor are only used in draw() so put those variables there
- colours are not strings?
- make arrays of the tulips and clouds
- put all displays+drives together (I believe if you use arrays you don't need to make functions either as you can display it in one line)
- in mousePressed(): pressed = true is unnecessary
- hard coded
- good variable names, maybe wouldn't have used drive but e.g. movement for clouds
- good subdivision of classes
- generally good comments

Feedback given by participant 3 during evaluation workshop 2

4: Workshop 3

4.1: Workshop 3 - diagrams

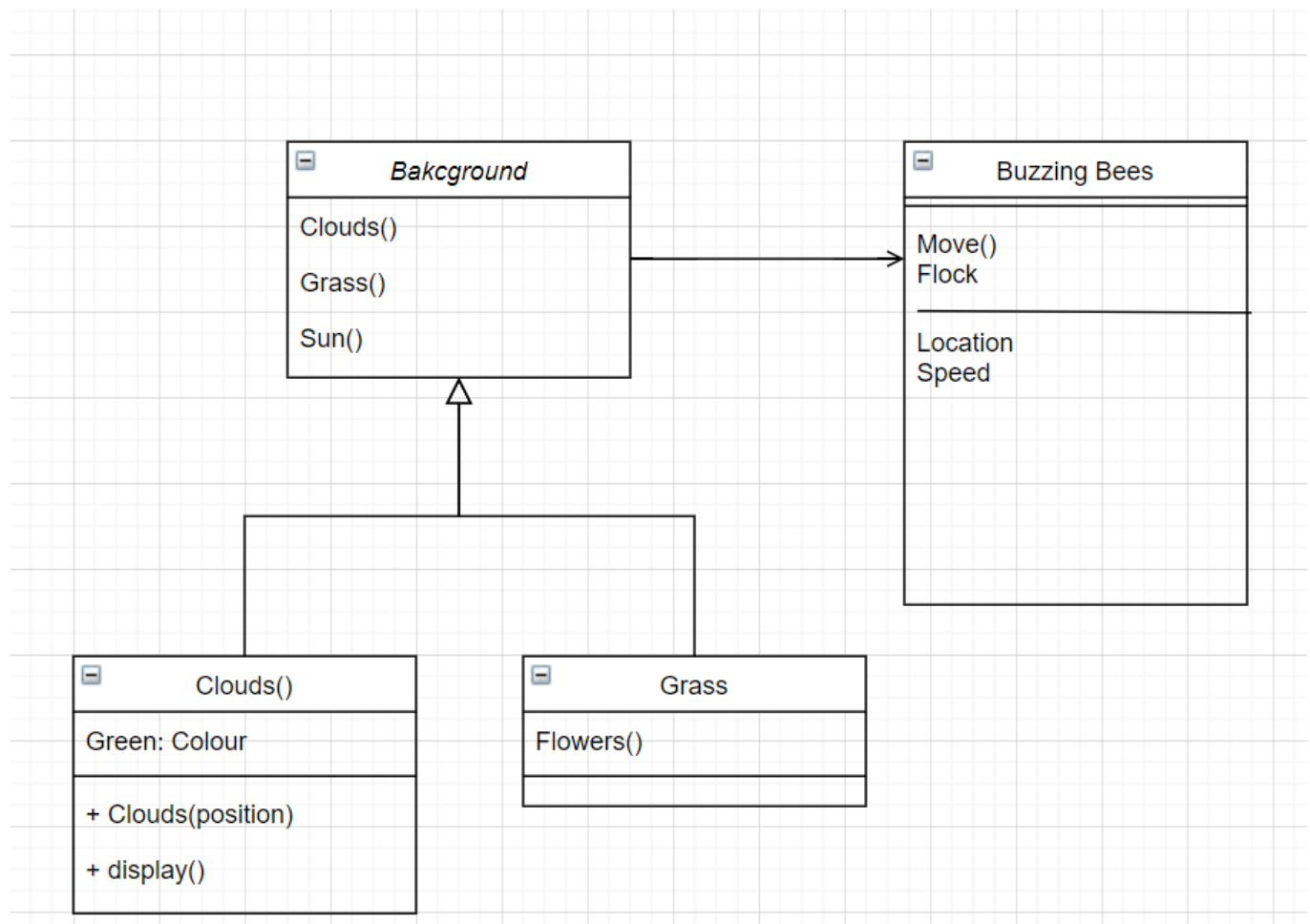


Figure 53: Class diagram made by participant 1 during evaluation workshop 3

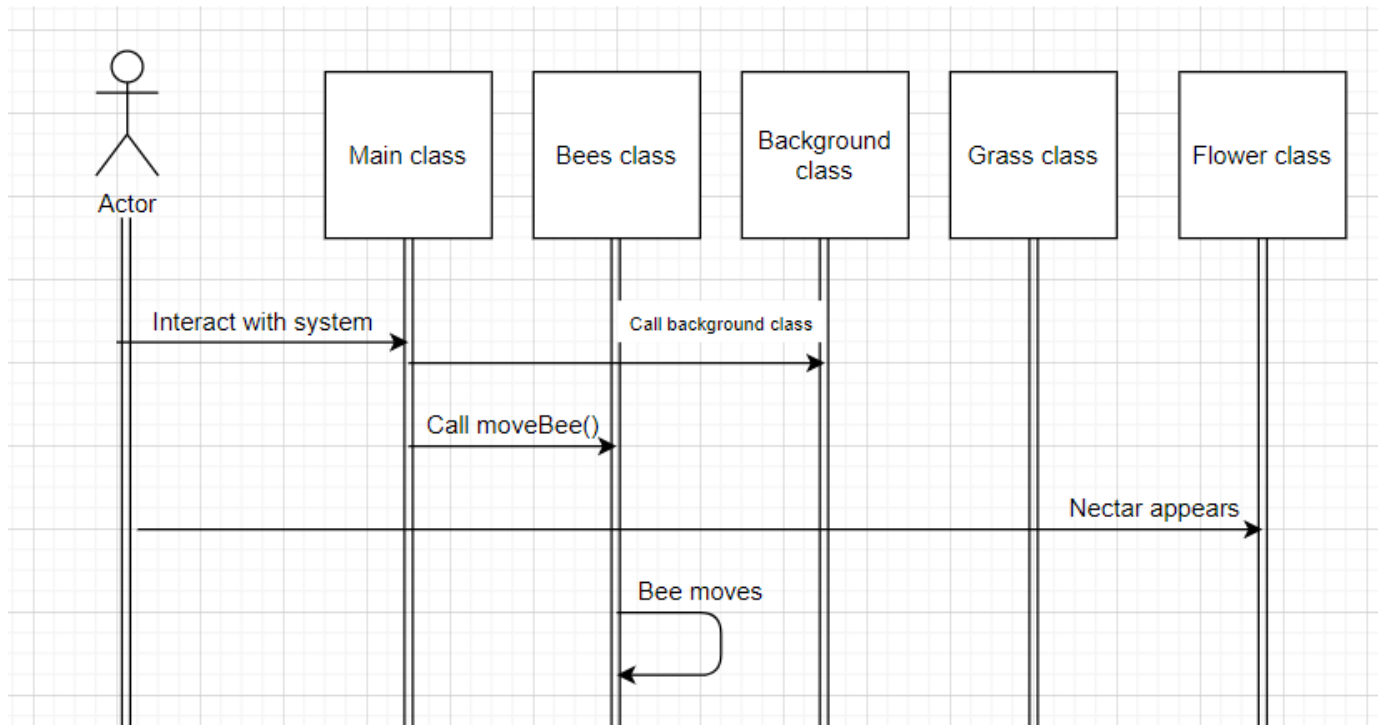


Figure 54: Interaction diagram made by participant 1 during evaluation workshop 3

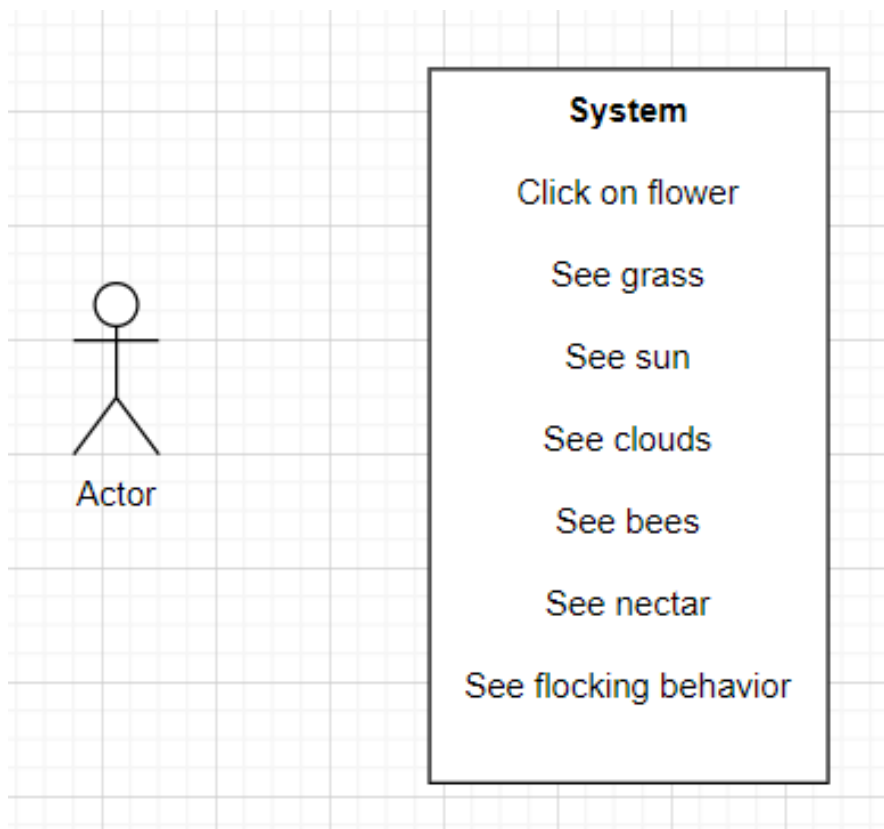


Figure 55: Use case diagram made by participant 1 during evaluation workshop 3

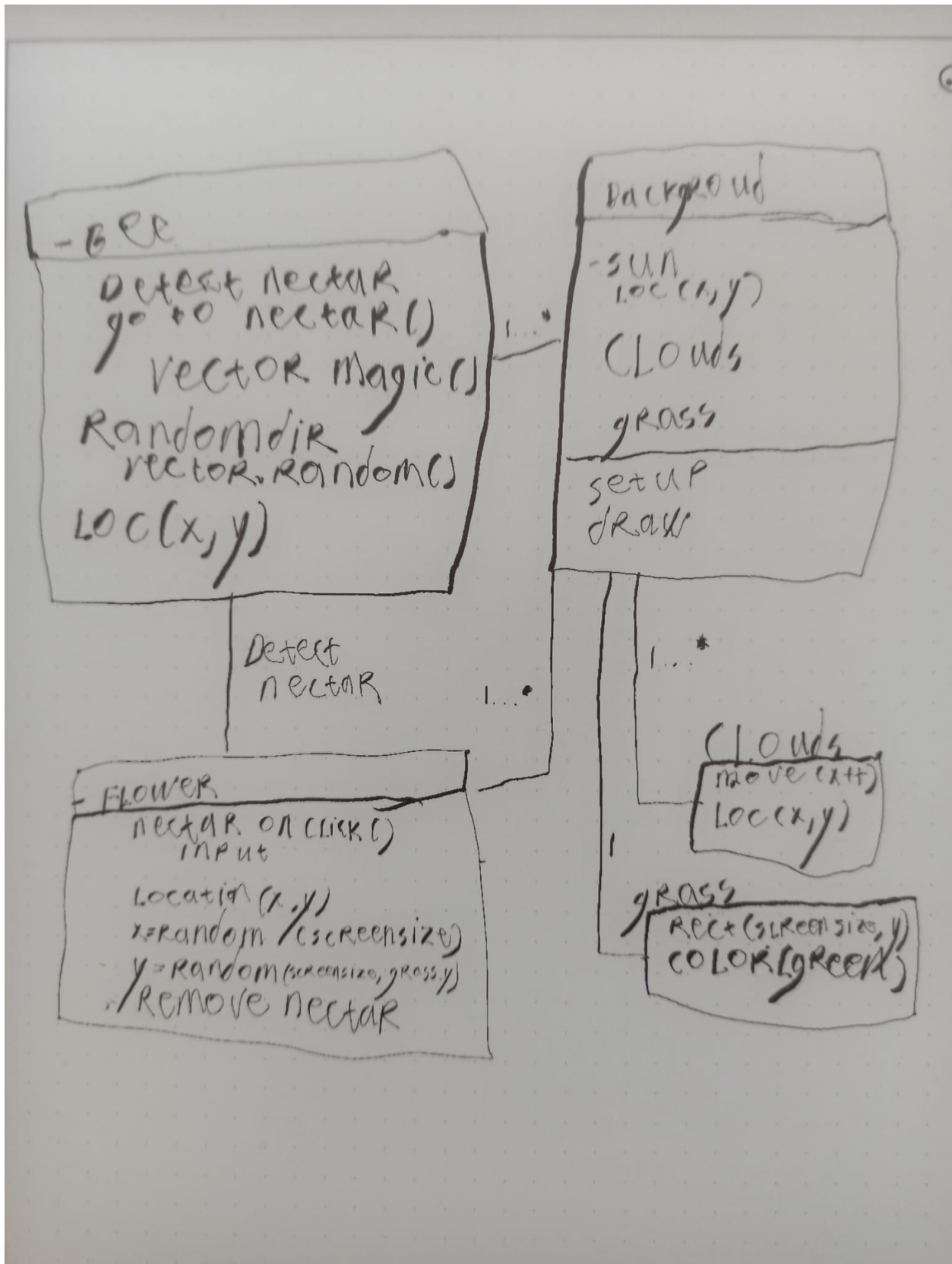


Figure 56: Class diagram made by participant 2 during evaluation workshop 3

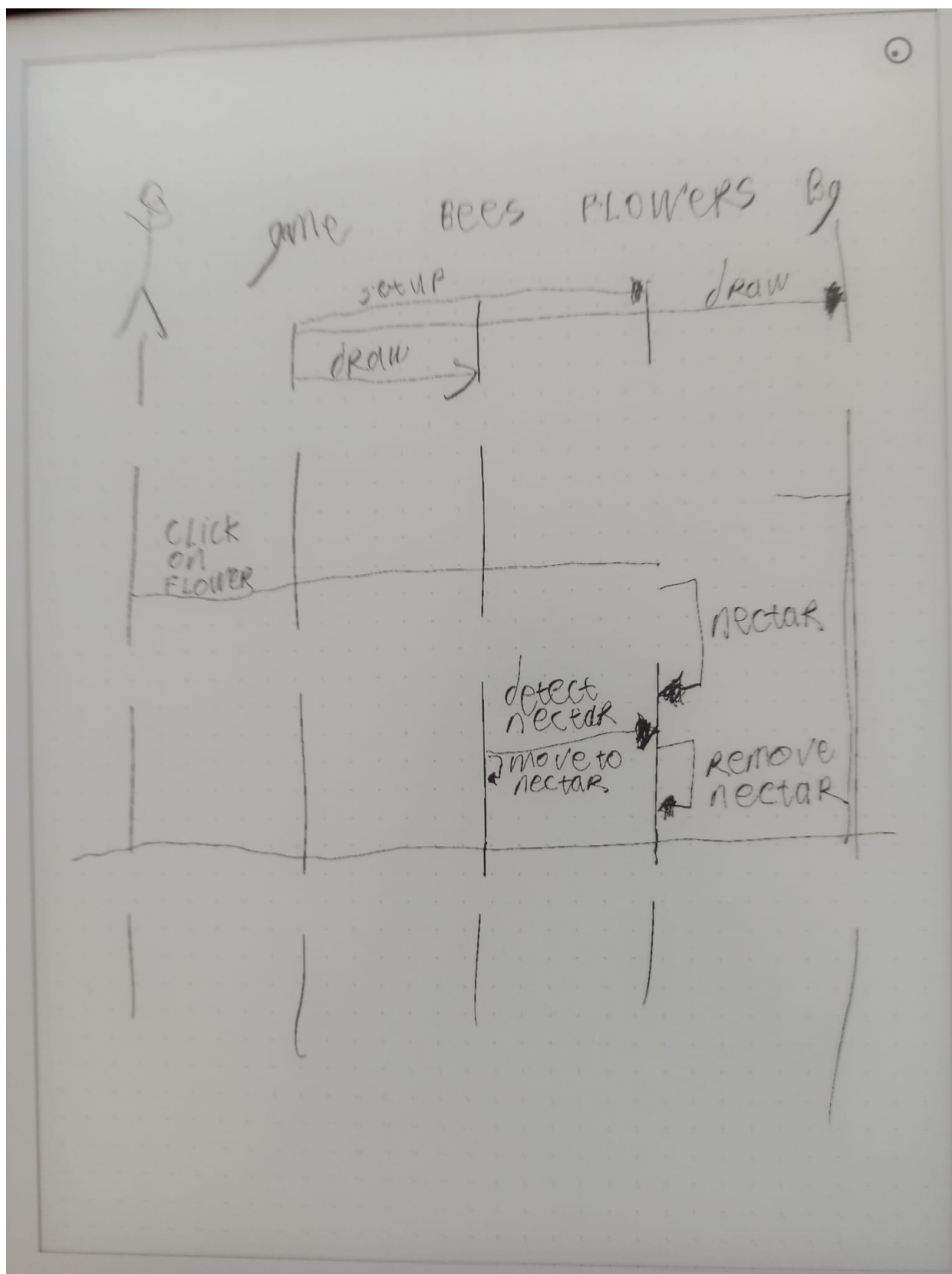


Figure 57: Interaction diagram made by participant 2 during evaluation workshop 3

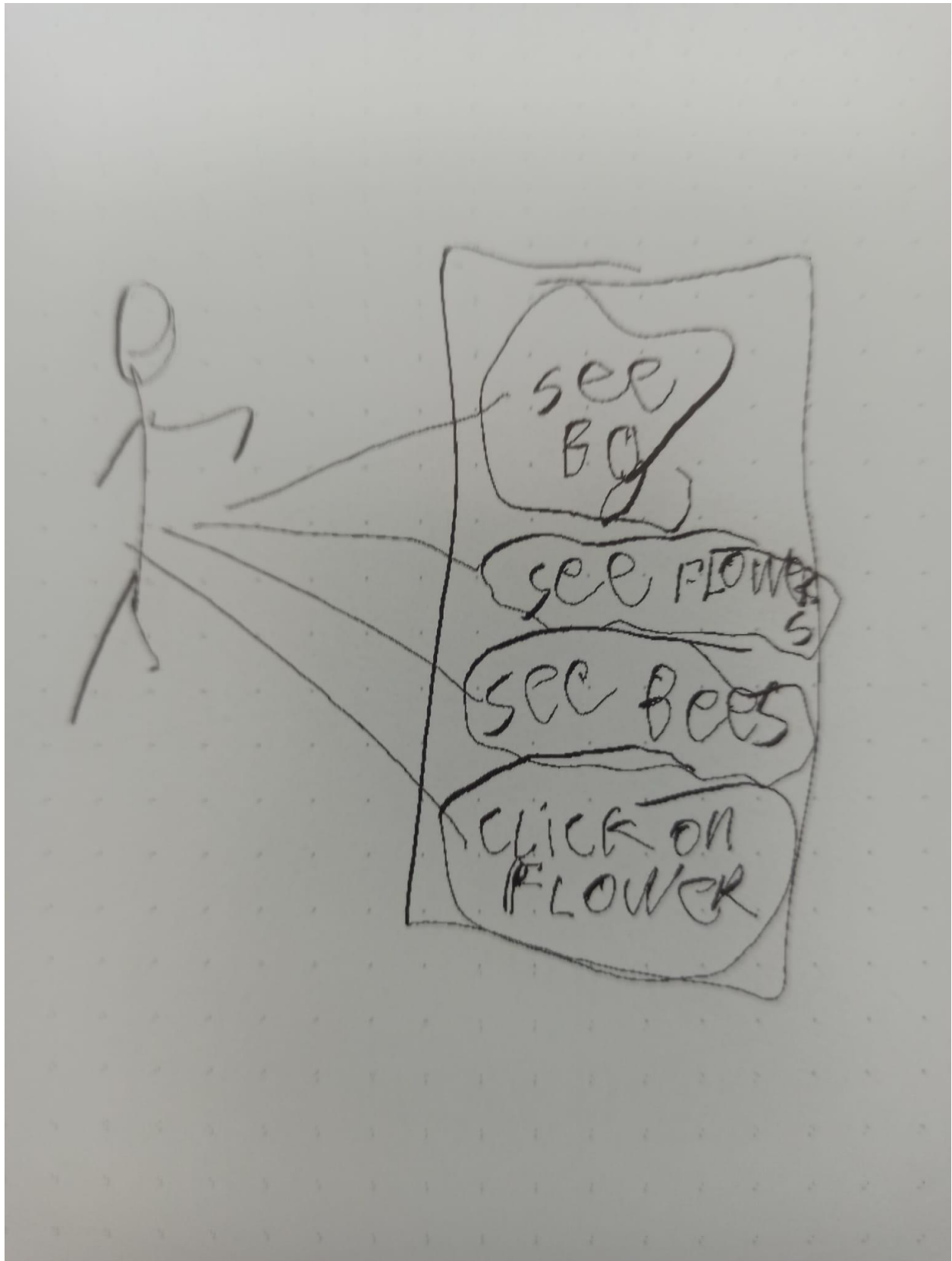


Figure 58: Use case diagram made by participant 2 during evaluation workshop 3

4.2: Workshop 3 - feedback

General feedback: In the sketch explanation you say that there are two yellow flowers and one red. In the output it is the other way around. Not sure if it is on purpose or not but I would double check that :)

Feedback code:

I would make an array of clouds instead of multiple clouds like you did now.

In general the comments look fine

The main class has too much in it. I would mainly just call functions from classes and do less in the main itself

Feedback given by participant 1 during evaluation workshop 3

Every cloud and flower is initiated separately. What happens if you need hundreds? Generate them instead of doing each one separately

displayFlowers and closeFlowers, are closed flowers invisible?

mousePressed and mouseReleased have open- and closeFlowers functions in them. Why?

All functions can do with a lot of redundancy.

Why are we using images for the colored flowers? Can't we just edit the color of a main image?

So much copy and paste. Please have a 'generator' function

Feedback given by participant 2 during evaluation workshop 3

Appendix 7: TA evaluation presentation slides



Figure 59: Slide 1 of the TA evaluation presentation

Introduction

- Why this workshop?
 - Observations within CreaTe
 - Programming is hard!
 - Programming as a problem-solving activity → Software design
- Workshop content
 - Groups of 2
 - 1.5 days
 - Module 3
 - Processing



Figure 60: Slide 2 of the TA evaluation presentation

Overall

- Is the workshop worth adding?
- What do you think of the overall structure?
- How much ECs should completing the workshop be worth?



Figure 61: Slide 3 of the TA evaluation presentation

Assignment 1

- Create a design (three diagrams) from a textual description
- 2 hours
- **TA involvement:**
 - Help with creating designs
 - Give feedback on designs
 - Check final results and approve or disapprove

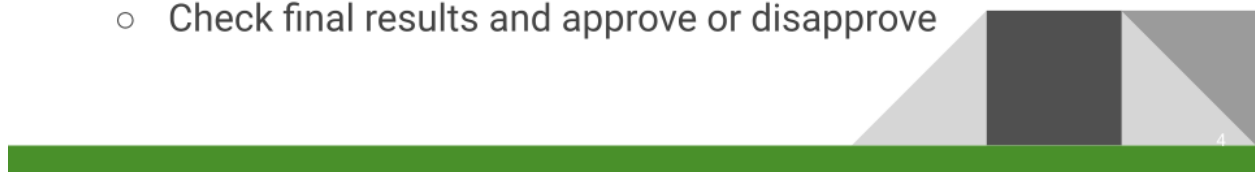


Figure 62: Slide 4 of the TA evaluation presentation

Assignment 2

- Create code based on own design
- 5 hours (+hours outside office hours)
- **TA involvement:**
 - Help students with their code
 - Provide feedback on code
 - Final approval is not needed



Figure 63: Slide 5 of the TA evaluation presentation

Assignment 3

- Provide feedback on received code and again create a design (3 diagrams)
- 4 hours
- **TA involvement:**
 - Help students with giving feedback and recommendations
 - Help with creating designs
 - Give feedback on designs
 - Check final results and approve or disapprove

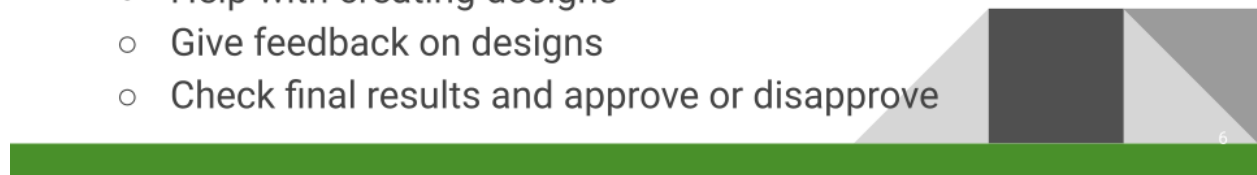


Figure 64: Slide 6 of the TA evaluation presentation

Additional workload

- Enough knowledge on each subject is needed
- Preparing assignment 3 (handing out code)
- Grading



Figure 65: Slide 7 of the TA evaluation presentation

Lectures

- Lecture 1.1: **Introduction to software design** (30 min)
 - Lecture 1.2: **Introduction to assignment 1** (15 min)
 - Lecture 2.1: **Introduction to assignment 2 and 3** (20 min)
 - Lecture 2.2: **Good coding habits and common mistakes** (40 min)
 - Lecture 2.3: **Introduction to Git** (30 min)
-
- Time spent
 - Order



Figure 66: Slide 8 of the TA evaluation presentation

Appendix 8: Workshop content

1: What is software design?

1.1: What is software design? - Presentation slides



Figure 67: Slide 1 of the “What is software design?” presentation

What is software design?

- ‘Design’ choices
- Abstractions on code level
- System behaviour



Figure 68: Slide 2 of the “What is software design?” presentation

Why is software design important (for CreaTe)?

- Understanding programming foundations
- Influence on outcome
- Performance, Maintainability
- Quality of code, time needed to fix bugs
- Understandability
- Asking targeted questions



Figure 69: Slide 3 of the “What is software design?” presentation

Software design tools

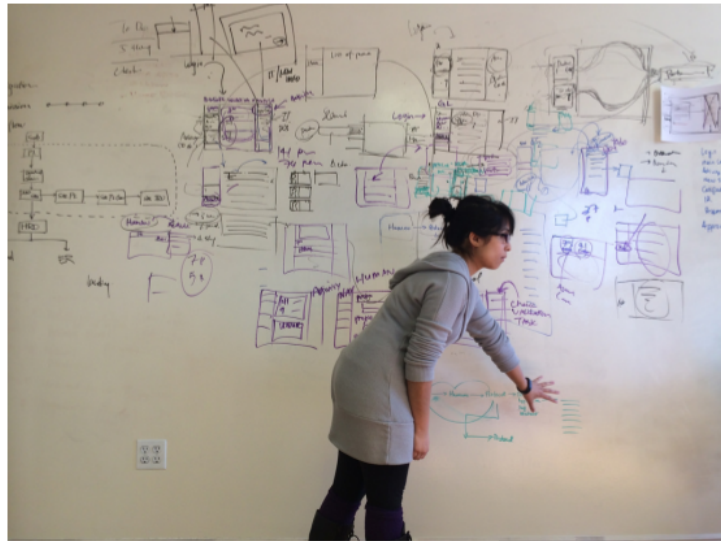
- Formal vs informal
- Formal: Unified Modelling Language (UML), UTML
- Informal: Photoshop, whiteboard, pen & paper, etc.
- Not mandatory to use UML for this workshop



Figure 70: Slide 4 of the “What is software design?” presentation

Real-life examples

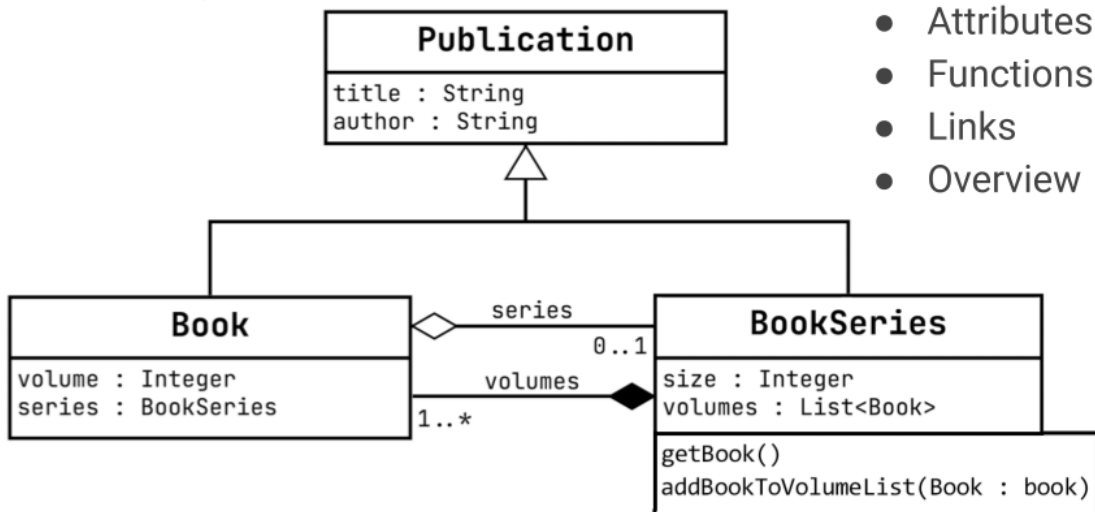
- Diagrams:
 - Class
 - Interaction
 - Use case



Source:
<https://github.com/foundersandcoders/Workshop-Software-Architecture-Design/blob/master/README.md>

Figure 71: Slide 5 of the “What is software design?” presentation

Class diagram



Source: UT TCS module 2, design course by Vadim Zaytsev

Figure 72: Slide 6 of the “What is software design?” presentation

Class diagram

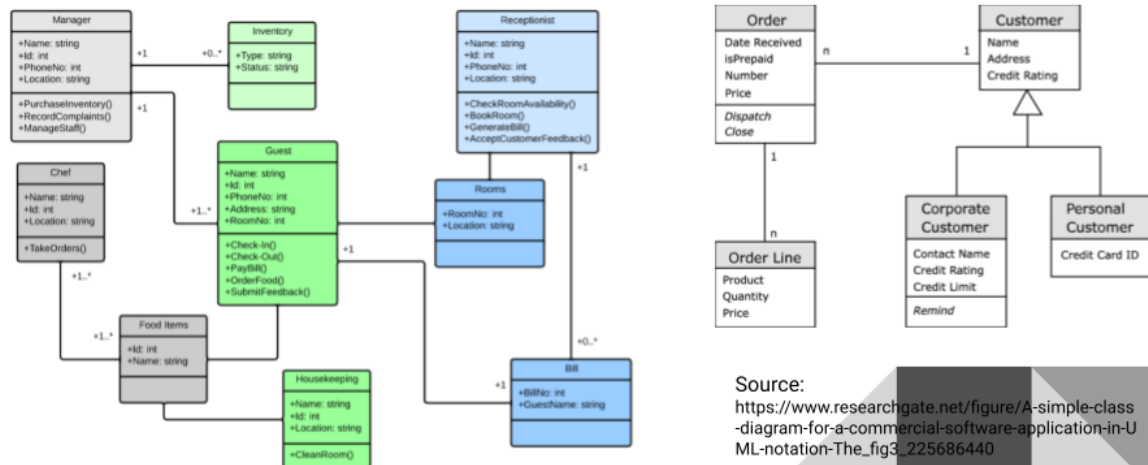


Figure 73: Slide 7 of the “What is software design?” presentation

Class diagram

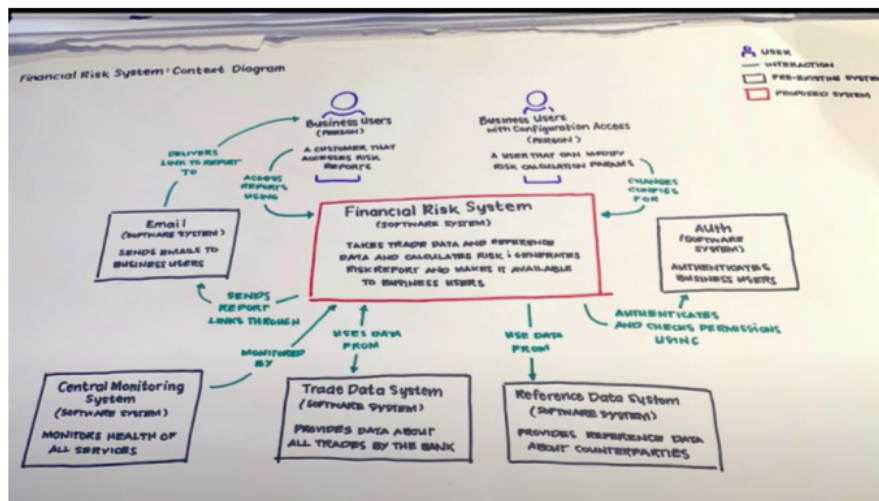
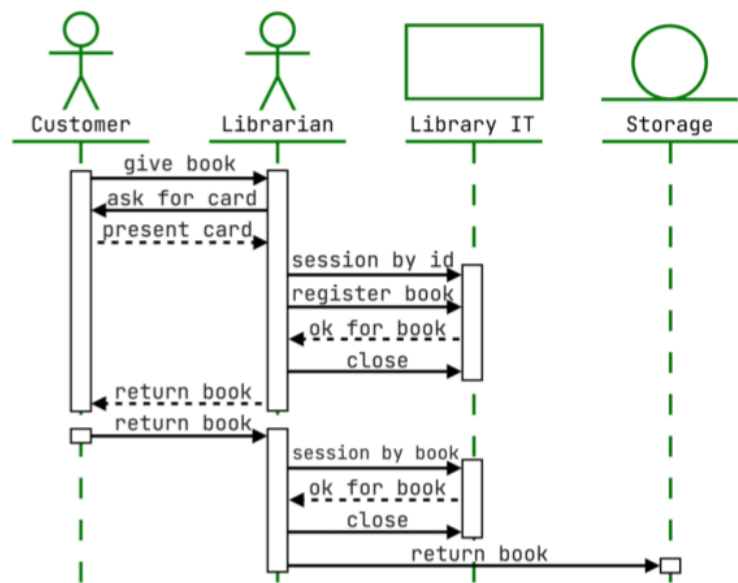


Figure 74: Slide 8 of the “What is software design?” presentation

Interaction diagram

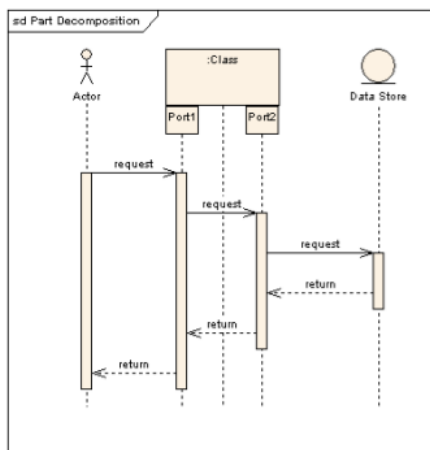
- Dynamic behaviour
- Different parts of the system
- Actors
- Interaction
- Accessible to non-coders
- Not code language dependent



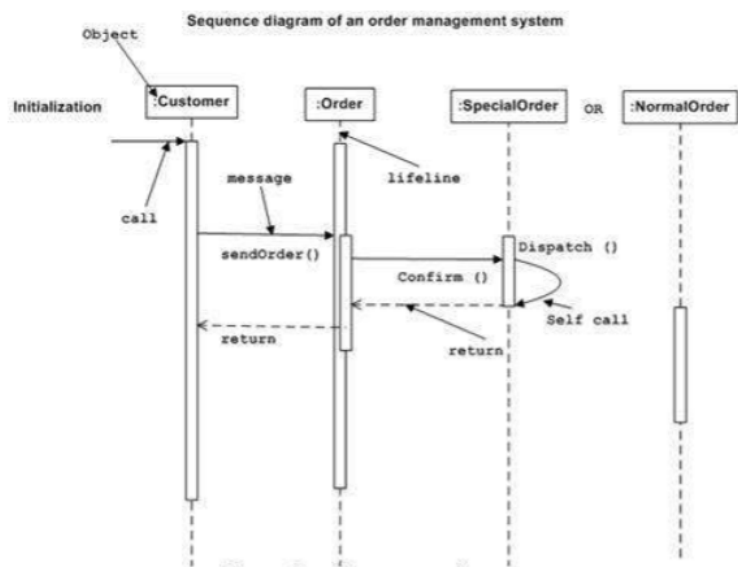
Source: UT TCS module 2, design course by Vadim Zaytsev

Figure 75: Slide 9 of the “What is software design?” presentation

Interaction diagram



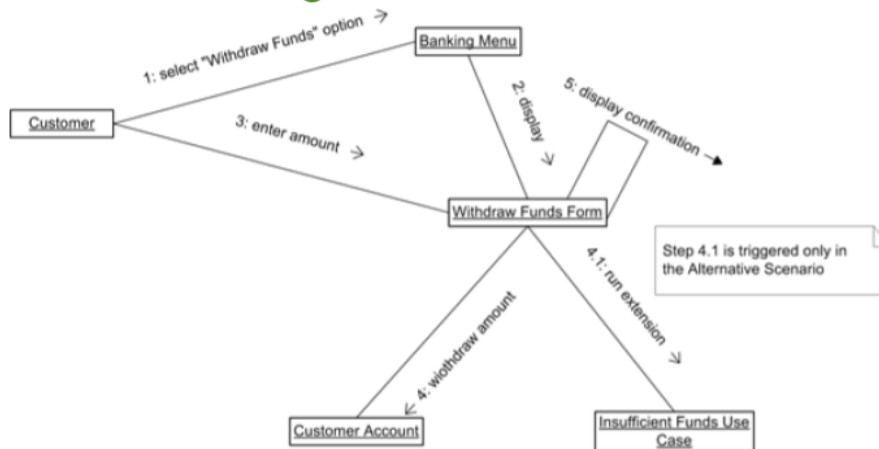
Source:
<https://sparxsystems.com/resources/tutorials/uml2/sequence-diagram.html>



Source:
<https://www.hebergementwebs.com/uml-tutorial/uml-interaction-diagrams>

Figure 76: Slide 10 of the “What is software design?” presentation

Interaction diagram



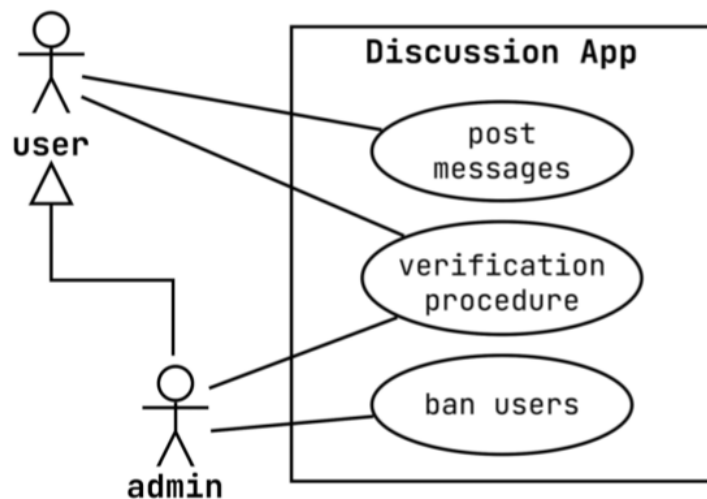
Source: <https://modernanalyst.com/Resources/Articles/tabid/115/ID/2018/End-to-End-UML-Sequence-Diagram.aspx>

11

Figure 77: Slide 11 of the “What is software design?” presentation

Use case diagram

- System
- Actors
- Use cases
- Links
- Checking requirements
- Easy to understand



Source: UT TCS module 2, design course by Vadim Zaytsev

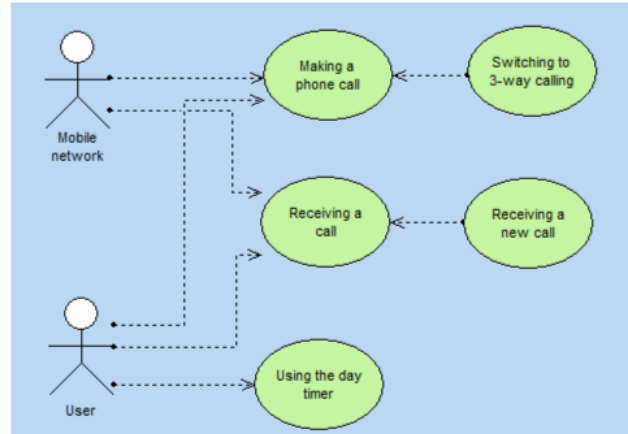
12

Figure 78: Slide 12 of the “What is software design?” presentation

Use case diagram



Source: <https://www.lucidchart.com/pages/uml-use-case-diagram>



Source:
https://help.windev.com/en-US/?2035004&name=use_case_diagram

13

Figure 79: Slide 13 of the “What is software design?” presentation

1.2: What is software design? - Content

What is software design?

As the name implies, software design is concerned with design choices related to software. In this case, design does not mean the chosen colours or layouts of the visuals produced by the code, but the design process of the code itself. Software design is responsible for choices made on code level, so which classes are needed, how the functionality is going to be implemented using different functions and how different pieces of code are related to each other. It shows how the system is organized and how it should behave [48].

Why is software design important?

There are quite some students within CreaTe who have never programmed before they joined the study. Software design can help with understanding the basic concepts of programming as a whole. This will, in combination with lectures on syntax they will receive during other courses, make it easier for students to understand the basic foundations of programming. These foundations are very important to understand as a student. Without, students would just write code without knowing what they are doing and these foundations can help with becoming better at programming more easily. Additionally, the way you design your code directly influences how the software will turn out. Design choices impact the quality of your code, as well as the performance, maintainability, understandability and time needed to fix bugs. By thinking about what you are going to write first, you create a mental model of the project. A daunting description of what needs to be done turns into a clear problem for which you are going to design the solution. This can help quite a bit in understanding what needs to be done and what

is needed to reach the intended outcome. Next to that, it can help in determining which parts of a project are harder for you, making it easier to ask targeted questions. Being able to ask targeted questions is very beneficial for both students and TAs. Students who can clearly describe where they are stuck are easier to help, saving time and preventing frustration [48].

Which tools are available to work with software design?

There are different tools available to make designs for software. These tools range from formal to very informal, and which one you use depends on who you are creating the design for. If you are working at a company, the chances that your design should be formal and follow certain rules is quite high. However, if the design is just for you or your project group, an informal design might be the better choice. In this lecture, the formal tool UML (Unified Modelling Language) will be discussed. It includes a standard way of depicting certain software design diagrams, as well as notations for relations within diagrams. This language is used by many companies and it is likely that you will encounter it at a future job. A tool that can be used for this that is free for students, is Visual Paradigm. Students from TCS are also working on a more user-friendly tool, called UTML. However, software design can also be used informally. Some of you might have already made a design, perhaps without even knowing it. Simply writing down the name for a class and which functions this class should get is already a design, as is a scheme filled with pseudocode. Tools like Photoshop, Paint, but also a whiteboard or pen and paper can be used for this. For this workshop, the usage of UML is not mandatory. The most important part is that the design is clear, for both you as well as someone who has never heard before of the project you are working on.

Including examples of real-life software designs

Some examples of UML diagrams will follow. For this workshop, three kinds of UML diagrams have been selected: abClass diagram, Interaction diagram and use case diagram. If they sound daunting to you, don't worry. The examples use UML, which is not necessary for this workshop. However, it is good to have seen these examples at least once, since it is used in real life quite often.

- **Class diagrams**

The following Figure shows a simple class diagram:

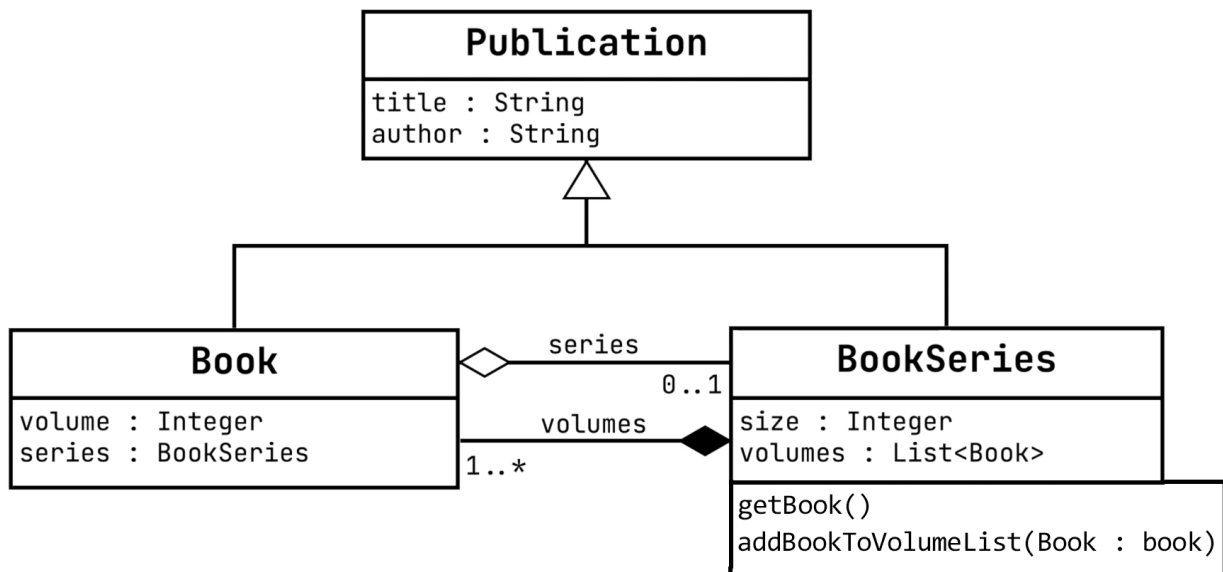


Figure 80: Class diagram example. [50]

Three classes can be distinguished: a 'Publication' class, a 'Book' class and a 'BookSeries' class. Additionally, attributes can be seen such as title and author in the Publication class, along with their types. In this case, both attributes are of type String. Different functions are also shown, such as `getBook()` and `addBookToVolumeList`, which takes a Book object as a parameter.

The links between the different classes can also be seen, in UML style.

- The white arrow indicates that both a Book and a BookSeries also have a title and an author: they inherit these attributes from this class. This was done because both these classes include these attributes, and it would be duplicate to include them in both classes.
- Next to that, a white arrow with 'series' goes from BookSeries to Book, with `0..1` next to it. This shows that a Book can be part of a BookSeries (1 if it does, 0 if it does not).
- Lastly, a black arrow with 'volumes' goes from Book to BookSeries with `1..*` next to it. This means that a BookSeries always contains at least one Book (or volume), which is stored in its volumes List. The star means that there could be an infinite number of books in a BookSeries.

Class diagrams can be used to get an overview of the classes and functions that are included in a system. They help form a mental model of your project as a whole, and can serve as a clear guideline on how to write your code.

- **Interaction diagrams**

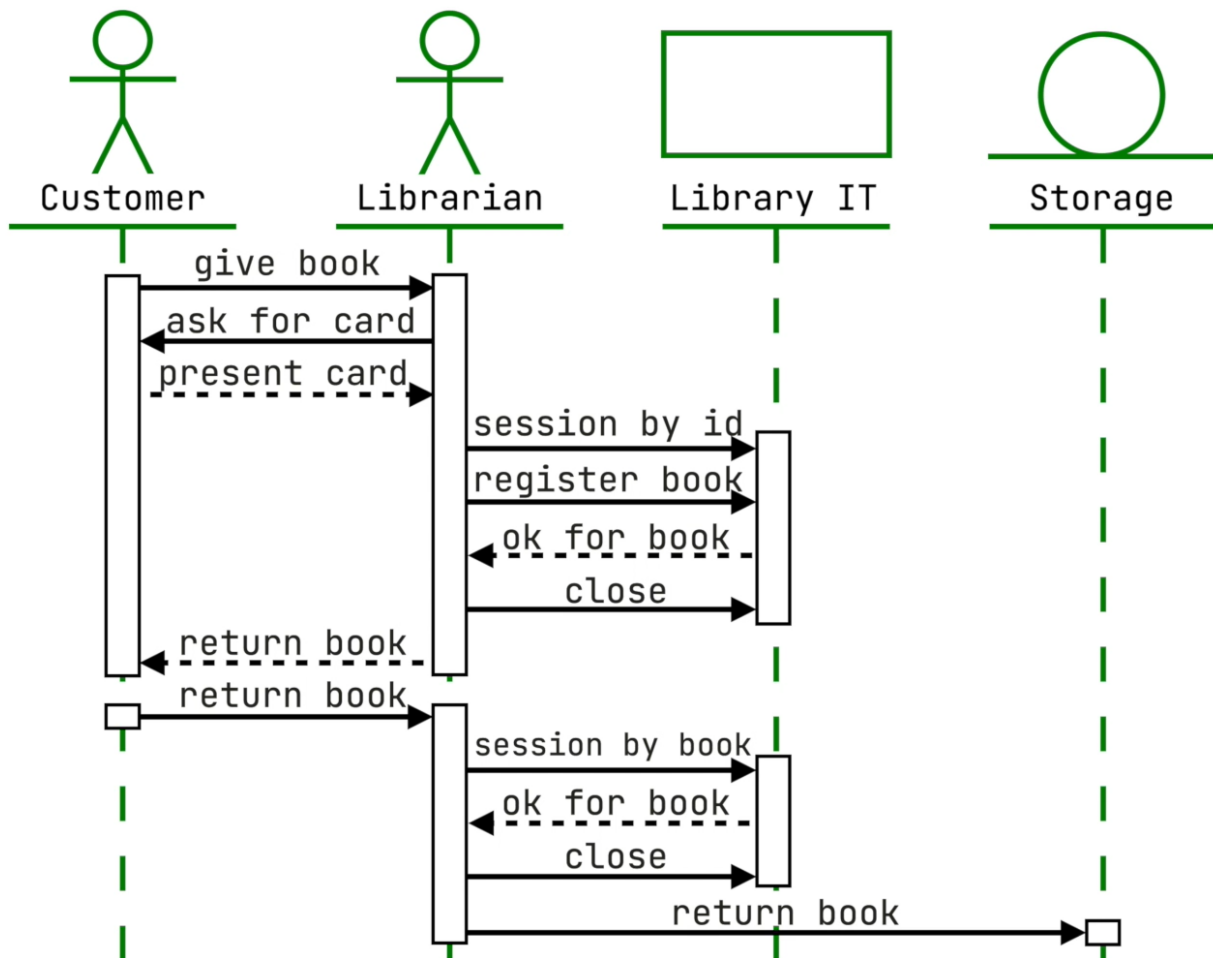


Figure 81: Interaction diagram example. [50]

With an interaction diagram, the different interactions between actors (stick figures) and different parts of the system (rectangle and circle) are displayed. The above Figure shows the process of a customer picking up and returning a book to the library. The librarian registers the book on the customer's account using the Library IT system. Once the customer returns the book, the librarian deregisters the book for the customer's account and returns the book to the storage.

Interaction diagrams are used to model the dynamic behaviour of a system, including the user interaction possibilities. They can show more clearly which bigger parts are included in a project and what impact a project will have on a more human level. Beneficial is that these kinds of diagrams are accessible to non-coders, since they do not include any code but plain text.

- Use case diagrams

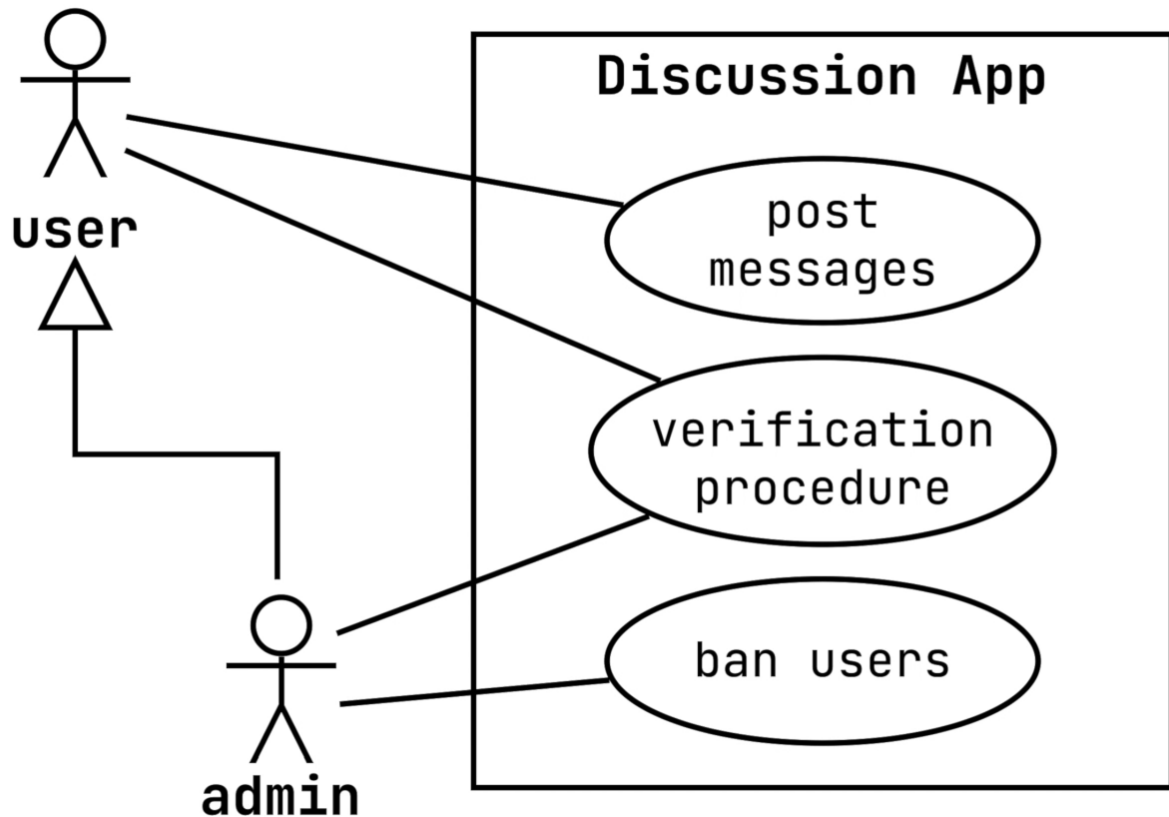


Figure 82: Use case diagram example. [50]

A use case diagram focuses on the actors within a system. The rectangle shows the system, the circles show which possible use cases there are for different actors. The above use case diagram depicts a Discussion App, where users can post messages and verify messages. An admin can also verify messages, as well as ban users. The arrow indicates that an admin inherits all use cases of a user.

A use case diagram can easily show if all the interaction requirements are in place. These kinds of diagrams are also easy to use for project teams, as they are easy to understand.

2: Assignment 1 introduction



Figure 83: Slide 1 of the “Assignment 1” presentation

From text to design

- You will receive a situation and textual description of the code
- The following should be included in your design:
 - Class diagram
 - Interaction diagram
 - Use case diagram
- The design can be made using an online tool (draw.io), application (Paint, Photoshop, Visual Paradigm) or offline (pen and paper, whiteboard)
 - When using an offline tool, make sure to make legible pictures

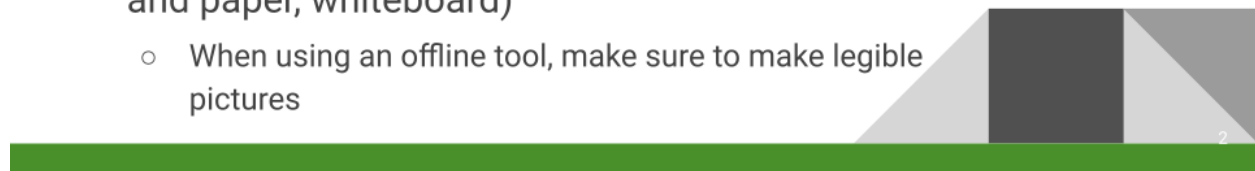


Figure 84: Slide 2 of the “Assignment 1” presentation

Example - description

Sketch: Croaking frogs

This sketch should contain a small game with green frogs in a pond. The user controls a fly: by using the arrow keys, the user can move the fly across the screen. If the fly comes too close to a frog, the frog will shoot its tongue towards the fly. If the fly does not move in time, it will be caught by the frog, and the user will lose the game. Above the screen, the amount of time the user has survived is shown. If the player is game over, the total amount of survived time will be shown in the middle of the screen and the counter will stop increasing. The user is given the option to start the game again.

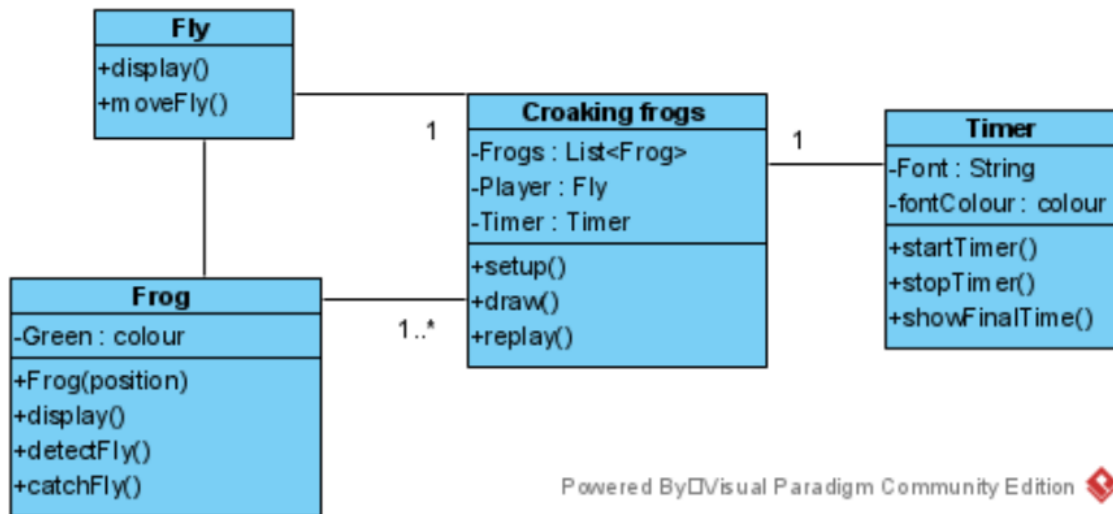
Figure 85: Slide 3 of the “Assignment 1” presentation

Example - elicited requirements

- A ‘frog’ class is needed for the frogs
- The frogs need to be green and shown on different locations
- Frogs should be able to detect a bypassing fly
- Frogs should be able to catch flies
- A ‘fly’ class is needed for the fly
- The fly should be able to fly based on user input
- A timer is needed to show the amount of survived time
- The timer should increase as long as the player is alive
- A main class is needed to create and draw the frogs, flies and background and deal with interaction
- The user should be able to see their score and replay the game

Figure 86: Slide 4 of the “Assignment 1” presentation

Example - class diagram

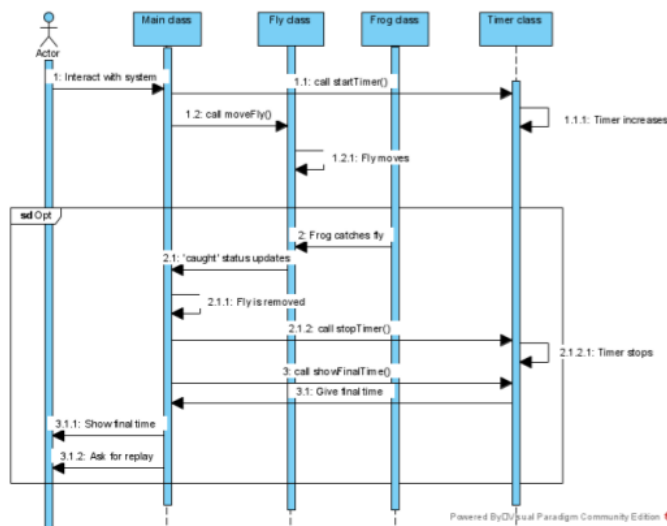


Powered By Visual Paradigm Community Edition

5

Figure 87: Slide 5 of the “Assignment 1” presentation

Example - interaction diagram



Powered By Visual Paradigm Community Edition

6

Figure 88: Slide 6 of the “Assignment 1” presentation

Example - use case diagram

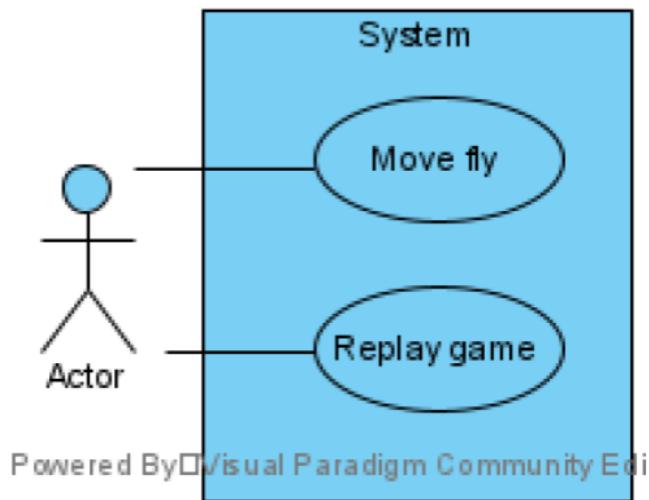


Figure 89: Slide 7 of the “Assignment 1” presentation

Keep in mind

- It is not mandatory to use correct UML
- Make sure the diagrams are understandable for both you and others
- The diagrams need to be checked by TAs
- A signoff is mandatory
- You have until the break (12:30)

Figure 90: Slide 8 of the “Assignment 1” presentation

3: Assignment 1

Software Design Workshop - Assignment 1

A museum in Enschede has an exhibition coming up, which focuses on Spring. The exhibition is meant for children up to 12 years old. There will be several different interactive installations that the children can use, each showing a different event associated with Spring. It is your task to write Processing sketches that portray the events and include the necessary interactivity.

Version A - Buzzing Bees

The sketch should portray bees in a flower field. The background of the sketch should show the sky with moving clouds, a green grass field, and a nice Spring sun. On the grass field, multiple flowers can be seen in random locations. The location, color, and type of flower vary. These attributes should be easily adjustable. Additionally, a beehive is located on the left side of the screen. Bees of different sizes should fly over the screen in random directions and with random speeds. If the user interacts with a flower, nectar appears on this flower. Bees flock towards this flower and collect the nectar. While there is nectar on one flower, the user cannot add nectar to another flower. After the bees have collected the nectar, the flock of bees returns to the beehive with the nectar. After some time, the bees exit the beehive and return to their previous behavior. Users should also be able to add more bees to the program by interacting with it. There should be a max of 50 bees on the screen at one time to make sure the system does not crash.

Version B - Windy Windmill

This sketch should portray a windmill in a grass field. The blades of the windmill should move, driven by wind. The background of the sketch should show the sky with moving clouds, a green grass field, and a nice Spring sun. The user should be able to spawn in three different types of goods that can go into the windmill: seeds, grain, and wood. These goods can be dragged towards the windmill. If the user drops one of the goods on top of the windmill while it is not already active, the windmill will start working. The user is shown a progress bar during this process. After the progress bar has reached its end, the user should receive a processed product: oil, flour, or chopped wood respectively. Users should also be able to increase the speed of the wind and thus the speed with which the windmill can process goods. It should be possible to delete any goods, processed or not, from the scene.

4: Assignment 2 & 3 introduction



Figure 91: Slide 1 of the “Assignment 2 & 3” presentation

From design to code

- Make a working sketch based on your own design
- All requirements mentioned in the textual description should be included in the final product
- Write the code as if you’re writing it for the museum staff: it should be understandable for people without a programming background



Figure 92: Slide 2 of the “Assignment 2 & 3” presentation

Keep in mind

- Ask a TA if you get stuck
 - Please note that TAs are only available during working hours
- No signoff is needed
- The code should be uploaded to Canvas under the name StudentNumber1_StudentNumber2_Version
 - Your version, A or B, can be found in the document you received of assignment 1
- You have until 11:30 tomorrow (doors open at 9:00)

Figure 93: Slide 3 of the “Assignment 2 & 3” presentation



Figure 94: Slide 4 of the “Assignment 2 & 3” presentation

From code to design

- You will receive code from another team
- This team has made code based on a different textual description
- The following needs to be provided:
 - A design including a class, interaction and use case diagram
 - A list with feedback and recommendations for the code



Figure 95: Slide 5 of the “Assignment 2 & 3” presentation

Keep in mind

- Keep your feedback constructive and clear
- The diagrams and list with feedback need to be checked by TAs
- A signoff is mandatory
- After you have signed off your work and discussed your feedback with the other group of students, you are free to go



Figure 96: Slide 6 of the “Assignment 2 & 3” presentation

5: Good habits and common programming mistakes

5.1: Good coding habits and common programming mistakes - Presentation slides

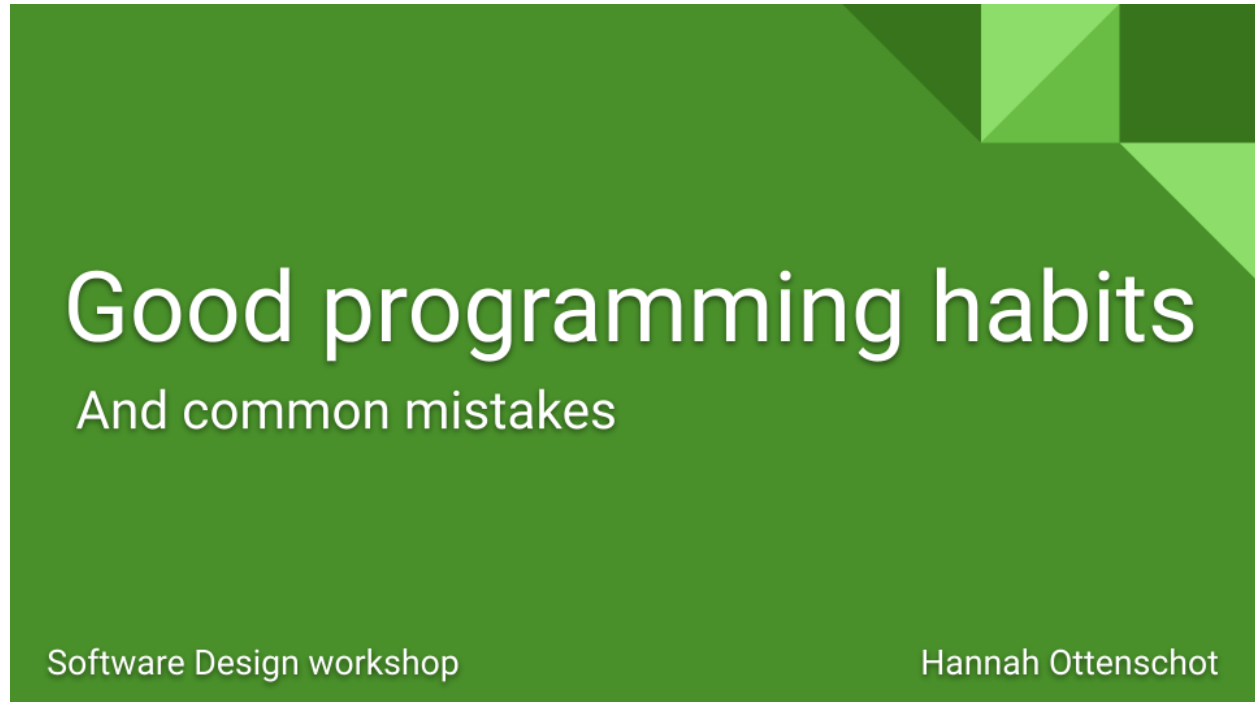


Figure 97: Slide 1 of the “Good coding habits and common programming mistakes” presentation

Why learn good programming habits?

- Become a better programmer
- More clarity for yourself
- More clarity for others
- Communication

```
func BrilliantFunction() (*Thing, error) {
    something, err := GetSomething()
    if err != nil {
        return nil, err
    }
    defer something.Close()
    if !something.OK() {
        return nil, errors.New("something not right")
    }
    another, err := something.Else()
    if err != nil {
        return nil, err
    }
    another.Lock()
    defer another.Unlock()
    err = another.Update(1)
    if err != nil {
        return nil, err
    }
}
```

Source:
<https://medium.com/@matryer/line-of-sight-in-code-186dd7cdea88>

Figure 98: Slide 2 of the “Good coding habits and common programming mistakes” presentation

Effects of good programming habits?

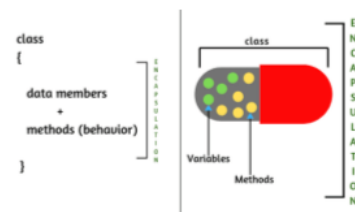
- Quality code
 - Clear to understand
 - Pleasant to work with
 - Easy to change
 - Watch out with hardcoding



Figure 99: Slide 3 of the “Good coding habits and common programming mistakes” presentation

General tips

- Keep your code clean
 - Comment where necessary
 - Note: comments do not compensate for bad code!
 - Remove dead code
 - Give your classes, variables and functions clear headers
- Use classes for separate concerns
 - Keep classes small
 - Isolated functionality is good
 - Nothing should do everything



Source:
<https://www.scientecheasy.com/2020/07/encapsulation-in-java.html/>

Figure 100: Slide 4 of the “Good coding habits and common programming mistakes” presentation

General tips

- Give meaningful names that are
 - Understandable
 - Revealing the intentions
 - Searchable
- Write code once
 - Do not copy code



Source: <https://www.whitesourcesoftware.com/resources/blog/copy-and-paste-code/>

5

Figure 101: Slide 5 of the “Good coding habits and common programming mistakes” presentation

General tips

- Write short methods
 - 10 - 20 lines
 - In practice: 95% contains 1 - 15 lines
- Write simple methods
 - Do not branch too much
 - Keeps methods easy to analyse, test and debug

```
public class BPMClassifier
{
    public static string Classify(int bpm)
    {
        if (bpm > HEART_RATE_ZONES.FAT_BURN_MIN)
        {
            if (bpm > HEART_RATE_ZONES.ENDURANCE_MIN)
            {
                if (bpm > HEART_RATE_ZONES.CARDIO_MIN)
                {
                    if (bpm > HEART_RATE_ZONES.THRESHOLD_MIN)
                    {
                        if (bpm > HEART_RATE_ZONES.PEAK_MIN)
                        {
                            if (bpm > HEART_RATE_ZONES.PEAK_MAX)
                            {
                                return "none";
                            }
                            return "peak";
                        }
                        return "threshold";
                    }
                    return "cardio";
                }
                return "endurance";
            }
            return "fat-burn";
        }
        return "none";
    }
}
```

When does this code return “cardio”?

Source:

<https://mattcbaker.com/posts/refactoring-nested-if-statements/>

6

Figure 102: Slide 6 of the “Good coding habits and common programming mistakes” presentation

5.2: Live coding session - content

The coding session should include:

- How to start up the IDE
- How to create a new working file
- Tips on where to begin
 - Determining requirements
 - Creating a design
 - Create a backbone first: start simple and add the complicated things later
- Creating of new classes
 - Tips on how to keep your class clean (naming, header)
 - Showing how Processing saves classes on your computer
 - Explanation of why certain things should go into certain classes (encapsulation)
 - Explanation of why global variables are generally not a good idea
 - Tips on how to keep the main class clean
- Writing methods
 - Tips on how to keep methods clean (naming, header, comments)
 - Example of bad use of hardcoding
 - Example of an overly complicated method
 - Example of an overly long method
 - Example of a getter and setter method
 - Example of a constructor

6: Introduction to Git

Initializing the repository and uploading the project

In this section, one team member is going to upload your project to GitLab. SNT provides all students with a free GitLab account which you can use as long as you are a student at the UT. The following steps have to be followed by only one team member.

1. Sign in at <https://git.snt.utwente.nl/snt> with your student account

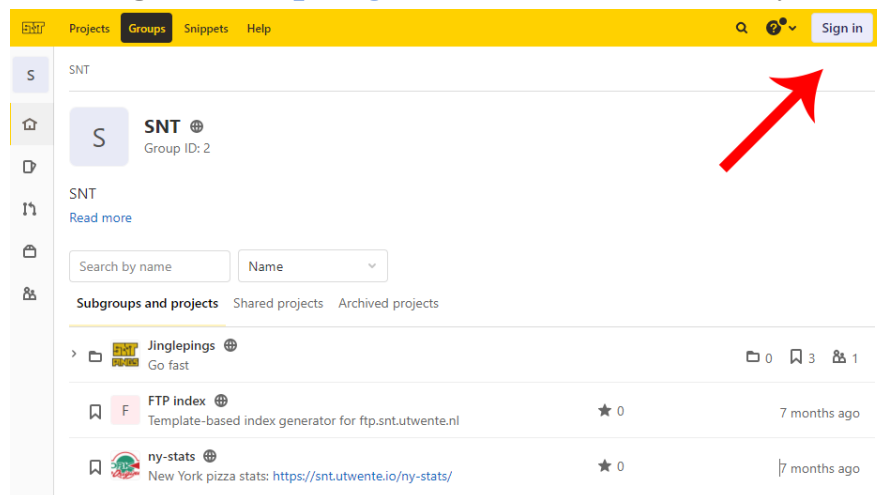


Figure 103: Location of the “Sign in” button for the Git introduction

2. Go to Projects in the top left corner

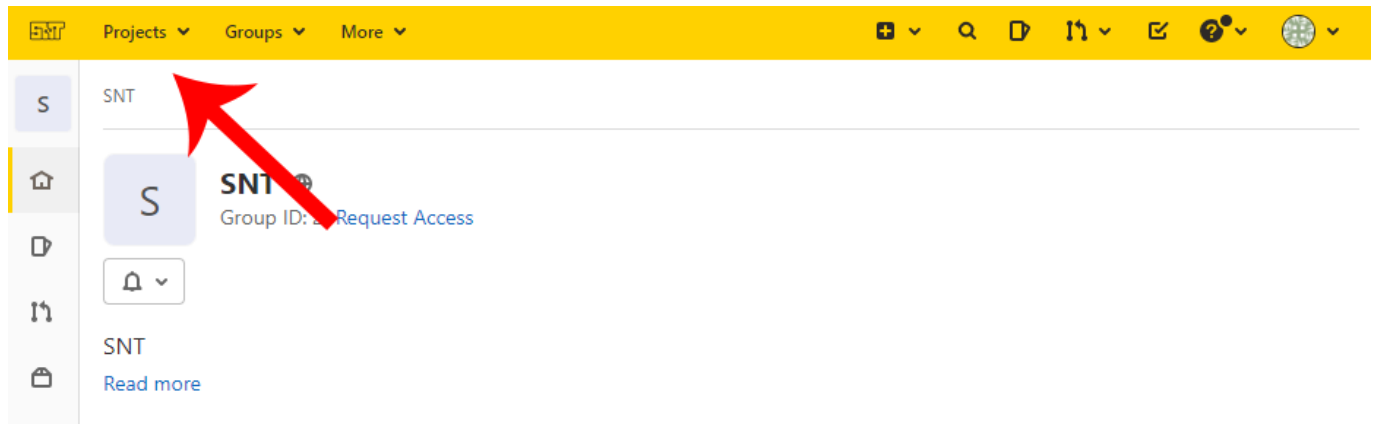


Figure 104: Location of the “Projects” button for the Git introduction

3. Go to Your projects

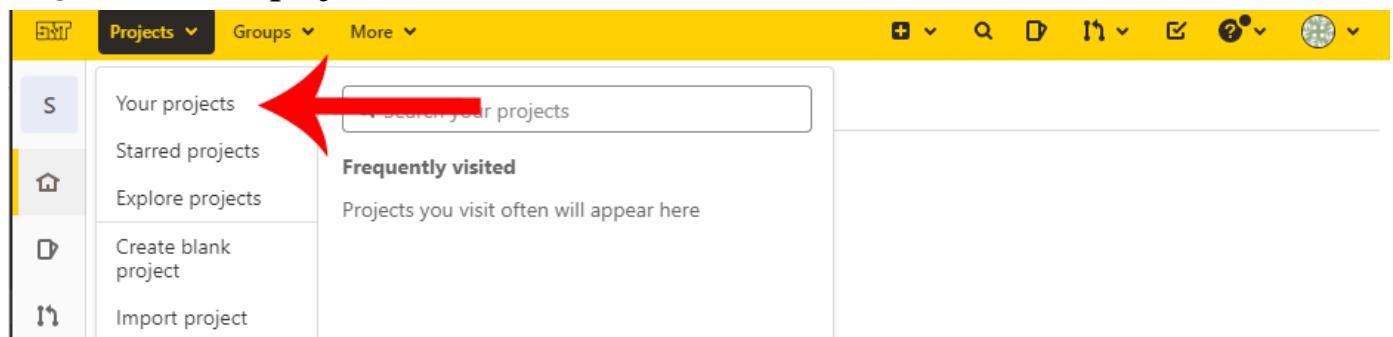


Figure 105: Location of the “Your projects” button for the Git introduction

4. Create a new blank project. Give the project a suitable name. A project description is optional. Make sure the project is private.

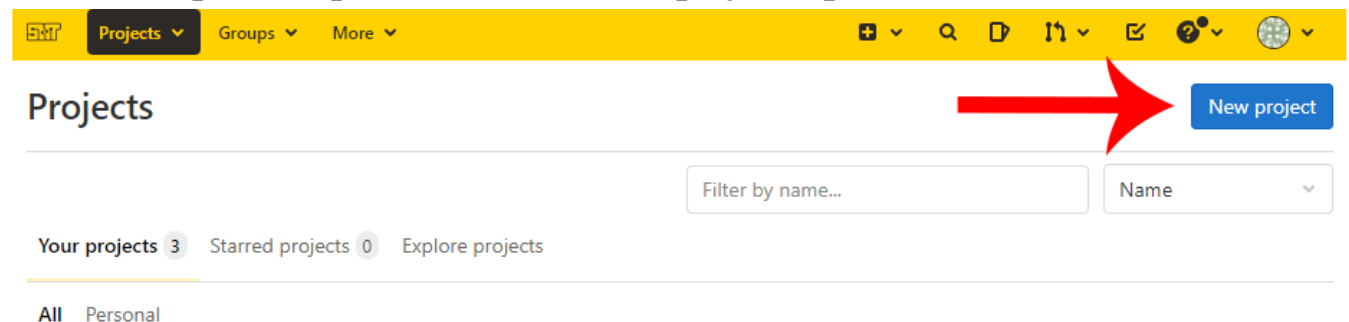
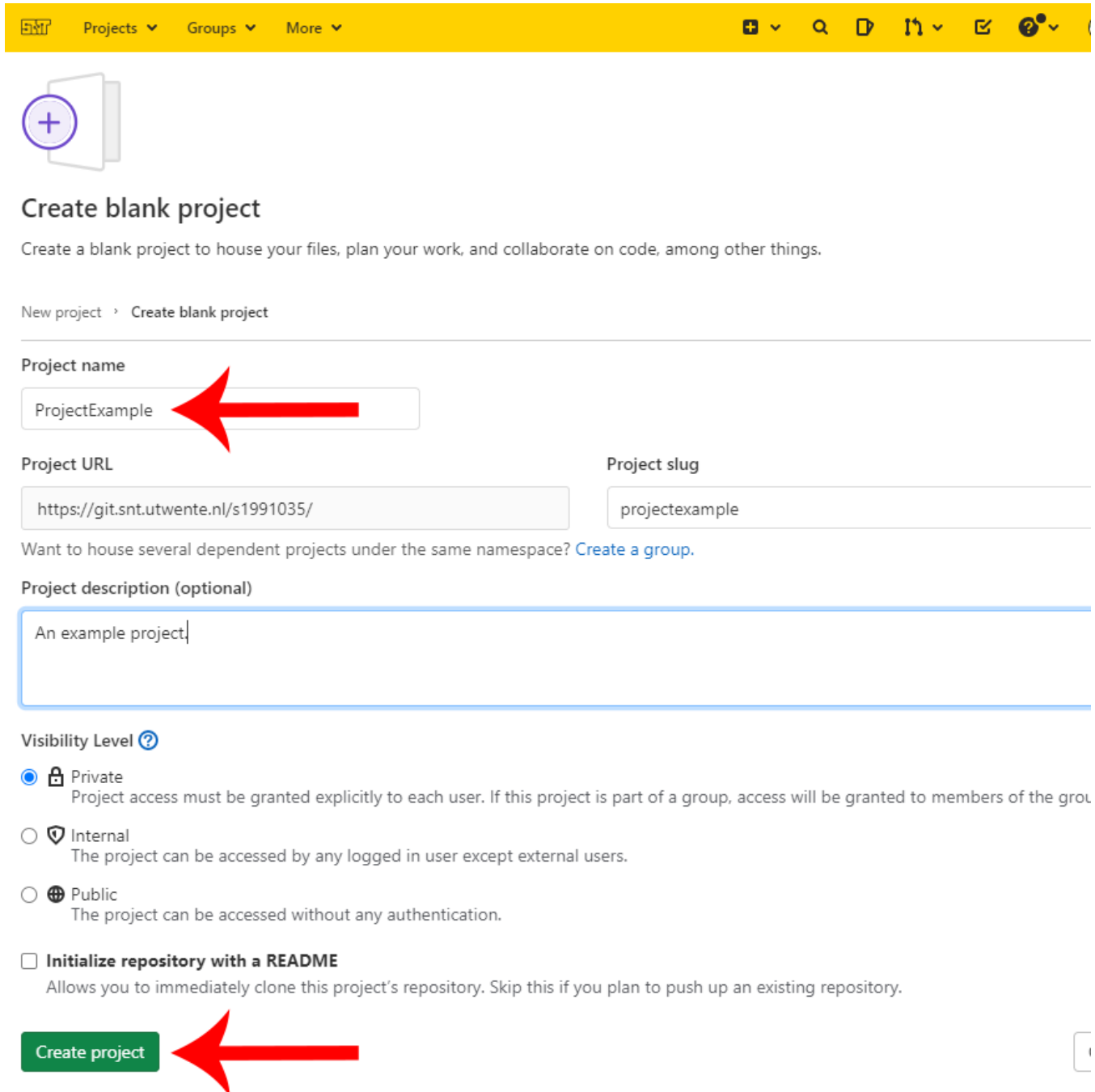


Figure 106: Location of the “New project” button for the Git introduction



Projects Groups More

Create blank project

Create a blank project to house your files, plan your work, and collaborate on code, among other things.

New project > Create blank project

Project name

ProjectExample

Project URL

https://git.snt.utwente.nl/s1991035/

Project slug

projectexample

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

An example project

Visibility Level ?

☒ Private
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

☐ Internal
The project can be accessed by any logged in user except external users.

☐ Public
The project can be accessed without any authentication.

☐ Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Figure 107: Location of the the “Project name” field and the “Create project” button for the Git introduction

5. After your project has been created, you will see your project page. A list with command line instructions is shown. This list will be handy later on.

Command line instructions

You can also upload existing files from your computer using the instructions below.

Git global setup

```
git config --global user.name "[REDACTED]"
git config --global user.email "[REDACTED]"
```

Create a new repository

```
git clone https://git.snt.utwente.nl/[REDACTED]/projectexample.git
cd projectexample
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Push an existing folder

```
cd existing_folder
git init
git remote add origin https://git.snt.utwente.nl/[REDACTED]/projectexample.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://git.snt.utwente.nl/[REDACTED]/projectexample.git
git push -u origin --all
git push -u origin --tags
```

Figure 108: SNT's Command line instructions for the Git introduction

6. **Find the location on your computer where the Processing project is saved.**
 - a. To get the location in the correct format, go to your file explorer and copy the location from there.
7. **Go to your computer's command prompt.**
 - a. On Windows, you can find this by typing "cmd" in your Windows search bar.
 - b. On Mac, you should look for "Terminal".
8. **In the command prompt, we need to navigate to where our project is saved. This can be done using the command "cd" followed by the location you determined in step 6.**
 - a. If your computer has multiple hard drives and you need to switch to another one, type in [Hard drive letter] followed by a :. For example, if you need to go to hard drive H, you need to type in H:.
9. **Once your command prompt is in the correct folder, it is time to register this folder as a Project folder and upload the content. This is where the list of command line instructions from the SNT page comes in handy. We will follow all the steps of the "Push an existing folder" section.**
 - a. First, type "**git init**". (We already did the step of "cd existng_folder". This will initialize an empty Git repository.
 - b. Then, we will register the SNT GitLab location as our repository. This is done with the command "**git remote add origin [https link of your project]**".
 - i. This HTTPS link is found by clicking on the Clone button, followed by the copy button of the "Clone with HTTPS".
 - c. Next is the command "**git add .**". This command tells Git you want to add a change in your working directory to the staging area. This means that Git now knows you want to make a change to particular files. In this case, the change is the uploading of your project.
 - d. Now we are going to commit. This is done by using the **commit** command.
 - i. Committing is like making a snapshot of the project.
 - ii. In the example of SNT, the commit command "**git commit -m "[commit message]"**" is used. The "-m "[commit message]" part could be replaced by a simple "-a" if you want to stage all commits. If you leave out this part and just use "**git commit**", a text editor is launched where a commit message can be written. This is bypassed by using "-m "[commit message]"
 - iii. To stick to the example, use the "**git commit -m "[commit message]"**" command. The message can be whatever you want.
 - e. Finally, it is time to push the changes to our repository. This is done by using the **push** command.
 - i. Pushing is the final step in the process and actually uploads the project to our remote repository.

- ii. In the example of SNT, the command “git push -u origin master” is used. The -u only has to be used the first time you are pushing something. It sets the local master branch on the remote under the name “origin” as the “master” branch to be the branch where you will push on. After you have done this once, you can leave out the -u part.
- iii. So, finish up by using the “**git push -u origin master**” command.

10. Congratulations, if all went well, you can now see your project on your project page of SNT’s GitLab!

- a. If you want to push any new changes, you can use the commands used earlier again:
 - i. “git add .”
 - ii. “git commit -m “[commit message]” “
 - iii. “git push origin master”

Pulling the project on another computer

It is time to make sure that both team members have access to the project repository. For this, you will need the computer of your team member. Or, they can follow these steps themselves of course! For clarity, the team member who followed the previous steps will be referred to as “student 1” and the other team member as “student 2”.

- 1. Make sure that student 2 signs in using their student account on <https://git.snt.utwente.nl/snt>.**
- 2. On the project page, student 1 can add student 2 to the project. Go to the Members tab.**
 - a. Here, you will see the Invite member section on top of the page.

Projects ▾ Groups ▾ More ▾

s1991035 > ProjectExample > Members

Project members

You can invite a new member to **ProjectExample** or invite another group.

Invite member

Invite group

GitLab member or Email address

Search for members to update or invite

Choose a role permission

Guest

[Read more about role permissions](#)

Access expiration date

Expiration date

Invite Import

Figure 109: Invite member section for the Git introduction

- b. The easiest way to find student 2 is by looking up their student number and typing in “@s[studentnumber]”.
 - c. Make sure to give them the Developer role. This will make sure that student 2 can make changes to the project.
 - d. The expiration date indicates when student 2 loses their rights to edit the project. This is optional.
 - e. Student 2 should now be able to see the project on their Your projects page.
3. **Student 2 can now clone the project to their computer.**
 - a. First, create a folder where you want to add the project in.
 - b. In the command prompt, navigate to this folder by using the “**cd [location]**” command.
 - c. Clone the project in this folder by using the “**git clone [HTTPS link]**” command.
4. **Student 2 can now make changes to the project and add, commit and push them!**
 - a. This can be done by using the commands from the previous section (subsection 10). When pushing the first time, make sure to include the “-u” part.

Merge conflicts

Git might have difficulty understanding what it should do when multiple developers of the project are trying to push their changes. In big projects, this is solved within the IDE (sadly, Processing does not have this feature) or by working with different branches within the repository. However, since the scope of this workshop is not that big, different branches might not be necessary. But, this does mean that you should be very careful with trying to push changes at the same time. Make sure that you communicate with your team member who is going to commit their changes when and try not to work on the same code at the same time.

7: Assignment 2

For the second assignment, students need to create code based on their own design. No extra slides are needed for this assignment. When students are working on this assignment, slide 3 of the Assignment 2 & 3 introduction presentation will be shown.

8: Assignment 3

For the third assignment, students will receive code from another group of students. Teachers and TAs will make sure the code ends up at the right team. This will be done via Canvas. No extra slides are needed for this assignment. When students are working on this assignment, slide 6 of the Assignment 2 & 3 introduction presentation will be shown.

References

- [1] M. K. C. Yeh, “Examining novice programmers’ software design strategies through verbal protocol analysis”, in *International Journal of Engineering Education*, 34, pp. 458–470, 2018.
- [2] M. Sperber, and M. Crestani, “Form over function: Teaching beginners how to construct programs”, in *Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP*, pp. 81– 89, 2012 <https://doi.org/10.1145/2661103.2661113>
- [3] F.E.V. Castro, S. Krishnamurthi, and K. Fisler, “The impact of a single lecture on program plans in first-year CS*”, in *ACM International Conference Proceeding Series*, pp. 118–122, 2017. <https://doi.org/10.1145/3141880.3141897>
- [4] M. Sperber and M. Crestani, “Form over function: Teaching beginners How To Construct Programs,” *Proc. ACM SIGPLAN Int. Conf. Funct. Program. ICFP*, pp. 81–89, 2012. doi: 10.1145/2661103.2661113.
- [5] M. Felleisen et al., “Program by Design”. Accessed on: March 03, 2021. [Online]. Available: <https://programbydesign.org/overview>
- [6] Burning glass technologies, “Beyond point and click - The expanding demand for coding skills”, Boston, Massachusetts, June 2016
- [7] M. Felleisen, R. B. Findler, M. Flatt and S. Krishnamurthi, “How to Design Programs, Second Edition”, MIT Press, 2014, Preface. Accessed on: March 04, 2021. [Online]. Available: https://htdp.org/2020-8-1/Book/part_preface.html
- [8] T. Y. Chen, R. McCartney, S. Cooper, and L. Schwartzman, “The (relative) importance of software design criteria”, in *Proc. 10th Annu. SIGCSE Conf. Innov. Technol. Comput. Sci. Educ.*, pp. 34–38, 2005, doi: 10.1145/1067445.1067458.
- [9] F. E. V. Castro, S. Krishnamurthi, and K. Fisler, “The impact of a single lecture on program plans in first-year CS*”, in *ACM Int. Conf. Proceeding Ser.*, pp. 118–122, 2017, doi: 10.1145/3141880.3141897.
- [10] E. Soloway, “Learning to program = learning to construct mechanisms and explanations”, in *Commun. ACM*, vol. 29, no. 9, pp. 850–858, 1986, doi: 10.1145/6592.6594.
- [11] F. E. V. G. Castro, “Investigating novice programmers’ plan composition strategies”, in *ICER 2015 - Proc. 2015 ACM Conf. Int. Comput. Educ. Res.*, pp. 249–250, 2015, doi: 10.1145/2787622.2787735.
- [12] M. De Raadt, M. Toleman, and R. Watson, “Training strategic problem solvers”, in *SIGCSE Bull. Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.*, vol. 36, no. 2, pp. 48–51, 2004, doi: 10.1145/1024338.1024370.
- [13] M. De Raadt, R. Watson, and M. Toleman, “Language Trends in Introductory Programming Courses”, in *Proceedings of the 2002 InSITE Conference*. 2002, doi: 10.28945/2464.

- [14] M. De Raadt, R. Watson, and M. Toleman, “Introductory Programming : What ’ s Happening Today and Will There Be Any Students to Teach Tomorrow ?,” ACE’04 Proc. Sixth Conf. Australas. Comput. Educ. Aust. Comput. Soc. Inc., pp. 1–6, 2004.
- [15] N. Wirth, “A Brief History of Software Engineering”, in IEEE Ann. Hist. Comput., vol. 30, no. 3, pp. 32–39, 2008, doi: 10.1109/MAHC.2008.33.
- [16] J. Warner, and P. J. Guo, “Examining how college hackathons are perceived by student attendees and non-Attendees”, in ICER 2017 - Proceedings of the 2017 ACM Conference on International Computing Education Research, pp. 254–262, 2017. <https://doi.org/10.1145/3105726.3106174>
- [17] J. Lave and E. Wenger, “Situated Learning: Legitimate Peripheral Participation.”, Cambridge University Press, 1991
- [18] M. Guzdial, “Learner-Centered Design of Computing Education: Research on Computing for Everyone”, in Synthesis Lectures on Human-Centered Informatics 8, 6, pp. 1–165, 2015.
- [19] A. Bandura, “Social Learning Theory”, Prentice Hall, 1977.
- [20] M.D. Sakhumuzi and O.K. Emmanuel, “Student perception of the contribution of Hackathon and collaborative learning approach on computer programming pass rate”, in 2017 Conference on Information Communication Technology and Society, ICTAS 2017 - Proceedings, pp. 1–5, 2017. <https://doi.org/10.1109/ICTAS.2017.7920524>
- [21] L.S. Vygotsky, “Mind in Society: The Development of Higher Psychological Processes”, Cambridge, MA: Harvard University Press, 1978.
- [22] X. Shaochun and V. Rajlich, “Pair programming in graduate software engineering course projects”, in Proceedings - Frontiers in Education Conference, FIE. vol. 2005, 1612027, Frontiers in Education - 35th Annual Conference 2005, FIE’ 05, Indianapolis, IN, United States, 19-22 October 2005.
- [23] G. Stahl, T. Koschmann and D. Suthers, “Computer-supported collaborative learning: A historical perspective”, In R. K. Sawyer (Ed.), Cambridge handbook of the learning sciences, pp. 409–426, 2006.
- [24] J. Vermillion and A. de Salvatierra, “Physical Computing, Prototyping, and Participatory Pedagogies Make-a-thon as interdisciplinary catalyst for bottom-up social change,” vol. 1, pp. 359–368, 2020, doi: 10.5151/proceedings-ecaadesigradi2019_593.
- [25] M. A. D. Soares, G. A. Silva, and T. F. P. Silva, “Make-a-thon : A blueprint for SDG-driven innovation Make-a-thon : A blueprint for SDG-driven innovation,” no. June, pp. 0–16, 2020.
- [26] <https://www.utwente.nl/en/education/bachelor/programmes/creative-technology/>
- [27] S. Hansen, N. Berente and K. Lyytinen, “Requirements in the 21st Century: Current Practice and Emerging Trends”, in Lyytinen K., Loucopoulos P., Mylopoulos J., Robinson B. (eds) Design Requirements Engineering: A Ten-Year Perspective. Lecture Notes in Business

Information Processing, vol 14. Springer, Berlin, Heidelberg, 2009.
https://doi.org/10.1007/978-3-540-92966-6_3

[28] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Int. J. Phytoremediation*, vol. 21, no. 1, pp. 137–172, 2003, doi: 10.1076/csed.13.2.137.14200.

[29] L. E. Winslow, "Programming pedagogy - A psychological overview," *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.)*, vol. 28, no. 3, pp. 17–23, 1996, doi: 10.1145/234867.234872.

[30] E. Soloway and E. Spohrer, "Novice mistakes: Are the folk wisdoms correct?", In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer*, pp. 401–416, 1989. Hillsdale, NJ: Lawrence Erlbaum.

[31] R.E. Brooks, "Towards a theory of the cognitive processes in computer programming", in *International Journal of Man-Machine Studies*, 9, pp. 737–751, 1977.

[32] R.E. Brooks, "Towards a theory of the comprehension of computer programs", in *International Journal of Man-Machine Studies*, 18, pp. 543–554, 1983.

[33] B. du Boulay, "Some difficulties of learning to program", In E. Soloway & J.C. Spohrer (Eds.), Hillsdale, NJ: Lawrence Erlbaum, pp. 283–299, 1989.

[34] C.L. Corritore and S. Wiedenbeck, "What do novices learn during program Comprehension?", *International Journal of Human-Computer Interaction*, 3, pp. 199–222, 1991.

[35] D. M. Kurland, R. D. Pea, C. Clement, and R. Mawby, "A Study of the Development of Programming Ability and Thinking Skills in High School Students," *J. Educ. Comput. Res.*, vol. 2, no. 4, pp. 429–458, 1986, doi: 10.2190/bkml-b1qv-kdn4-8ulh.

[36] D.N. Perkins and F. Martin, "Fragile knowledge and neglected strategies in novice Programmers", In E. Soloway & S. Iyengar (Eds.), *Empirical studies of programmers*, First Workshop, Norwood, NJ: Ablex, pp. 213–229, 1986.

[37] J.C. Spohrer, E. Soloway, E. and E. Pope, 'A goal/plan analysis of buggy Pascal programs', In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer*, Hillsdale, NJ: Lawrence Erlbaum, pp. 355–399, 1989.

[38] D.N. Perkins, C. Hancock, R. Hobbs, F. Martin, and R. Simmons, "Conditions of learning in novice programmers", In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer*, Hillsdale, NJ: Lawrence Erlbaum, pp. 261–279, 1989.

[39] M.C. Linn, and J. Dalbey, "Cognitive consequences of programming instruction", In E. Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer*, Hillsdale, NJ: Lawrence Erlbaum, pp. 57–81, 1989.

[40] D. M. Kurland, R. D. Pea, C. Clement, and R. Mawby, "A Study of the Development of Programming Ability and Thinking Skills in High School Students," *J. Educ. Comput. Res.*, vol. 2, no. 4, pp. 429–458, 1986, doi: 10.2190/bkml-b1qv-kdn4-8ulh.

- [41]** M. J. Van Gorp and S. Grissom, “An Empirical Evaluation of Using Constructive Classroom Activities to Teach Introductory Programming”, in *Computer Science Education*, 11:3, pp. 247-260, 2001. doi: 10.1076/csed.11.3.247.3837
- [42]** L. Williams, E. Wiebe, K. Yang, M. Ferzli and C. Miller, “In Support of Pair Programming in the Introductory Computer Science Course”, in *Computer Science Education*, 12:3, pp. 197-212, 2002. doi: 10.1076/csed.12.3.197.8618
- [43]** C. E. Wills, D. Deremer, R. A. McCauley and L. Null, “Studying the Use of Peer Learning in the Introductory Computer Science Curriculum”, in *Computer Science Education*, 9:2, pp. 71-88, 1999. doi: 10.1076/csed.9.2.71.3811
- [44]** J.R. Anderson, “Language, memory and thought”, Hillsdale, NJ: Erlbaum Associates, 1976.
- [45]** J.R. Anderson, “The architecture of cognition”, Cambridge, MA: Harvard University Press, 1983.
- [46]** J.R. Anderson, “Rules of the mind”, Hillsdale, NJ: Erlbaum, 1993.
- [47]** https://en.wikipedia.org/wiki/Dunning%E2%80%93Kruger_effect
- [48]** M. Jaiswal, “Software Architecture and Software Design,” *SSRN Electron. J.*, 2021, doi: 10.2139/ssrn.3772387.
- [49]** R. Wieringa, “Design Science Research Methods”, 2016. [Online] Available at: <https://wwwhome.ewi.utwente.nl/~roelw/DSM90minutes.pdf>
- [50]** V. Zaytsev, *Software Design*, course in Software Systems in academic year 2020-2021 (module 2 of Technical Computer Science and University of Twente)