Viggo Skarby

Fixing the Error

Debugging in Norwegian Computing Education

Master's thesis in Natural Science with Teacher Education
Supervisor: Monica Divitini
June 2022

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Viggo Skarby

Fixing the Error

Debugging in Norwegian Computing Education

Master’s thesis in Natural Science with Teacher Education
Supervisor: Monica Divitini
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Computing for everyone is a concept that is prevalent in the pedagogical environment today. This can in part be seen in the renewal of the Norwegian National Curriculum introduced in 2020. In the new curriculum there is an explicit focus on programming, both as a part of mandatory subjects and as computing electives. A central part of programming, and computing in general, is debugging. Debugging is the process of detecting, finding and fixing bugs in computer programs. Even though it is a process professional developers can spend up to half of their time on, debugging is reported to be an underrepresented part of the provided computing education at both tertiary and secondary education levels on multiple countries. Debugging also proves a hurdle in the K-12 classroom, and is a source of frustration for pupils when it comes to programming. There is also a lack of best practices for the debugging process and for the teaching of debugging in the classroom. The aim of this research project is to create an understanding of how debugging can be integrated in Norwegian computing education, by finding what the required skills and knowledge are and what the conditions and constraints for the development of teachers' debugging skills and didactics are. The foci of the research project is debugging knowledge and skills and debugging didactics from the perspective of educators of programming in Norwegian secondary school. To study this phenomenon, design science research has been conducted where learning objectives for teachers have been designed. The project has gone through 3 design cycles: 1) design of initial learning objectives, 2) expert evaluation and 3) teacher evaluation. The three cycles resulted in 5 learning objectives for teacher development of debugging skills and didactics. The learning objectives designed are A) knowledge on different types of bugs, B) how to use debugging strategies, C) how to facilitate a classroom culture for debugging, D) how to promote self-reliance and problem-solving in pupils' debugging and E) how digital tools may aid the debugging process. The main constraints for teachers development of debugging skills are lack of time and lack of learning material. To address the conditions and constraints learning material based on the learning objectives should be specific to subject curriculum and offer material that is close to the teacher's practice in their computing classroom.

Samandrag

Tanken om at alle skal lære om programmering og andre datarelaterte fagfelt er synleg i dagens pedagogiske utvikling. Vi ser det mellom anna i Fagfornyinga av det norske læreplanverket som kom i 2020, Kompetanseløftet 2020. Det nye læreplanverket har eit eksplisitt fokus på programmering, både i fellesfaga og i eigne valfag. Ein viktig del av programmering er feilsøking, som er prosessen av å oppdage, finne og rette opp feil i dataprogram. Sjølv om feilsøking er ein prosess som utøvande programmerarar og utviklarar kan bruke opp til halvparten av tida si på, er feilsøking ein underrepresentert del av programmeringsundervisinga. Dette gjeld både i grunnopplæringa og ved høgare utdanning i opptil fleire land. Feilsøking er eit hinder i klasserommet, og fører til at elevar blir frustrerte når dei skal programmere. Til tross for dette finst det ingen vedtekne retningslinjer for korleis ein burde drive verken feilsøking eller undervising av feilsøking. Dette forskingsprosjektet har som mål å skape forståing for korleis feilsøking kan integrerast i norsk programmeringsundervising. For å nå målet vil vi sjå på kva ferdigheiter og kunnskap ein må ha, og kva mogleggjerande eller grensande faktorar som finst, for at lærarar skal kunne utvikle sin feilsøking og feilsøkingsdidaktikk. Fokuset i forskingsprosjektet er ferdigheiter og kunnskap om feilsøking, i tillegg til feilsøkingsdidaktikk frå programmeringsundervisaren sitt perspektiv. For å undersøke fenomenet har læringsmål for undervisarar blitt designa gjennom *design science research*. Prosjektet består av 3 rundar med design: 1) design av første utkast av læringsmål, 2) ekspertevaluering og 3) lærarevaluering. Etter dei tre designrundane var resultatet 5 læringsmål for lærarar si utvikling av feilsøkingsferdigheiter og -didaktikk. Læringsmåla er A) kunnskap om ulike typar programfeil, B) korleis bruke feilsøkingsstrategiar, C) korleis fremje ein klasseromskultur for feilsøking, D) korleis fremje sjølvstendigheit og problemløysing i elevar si feilsøking og E) korleis digitale verktøy kan gje stø i feilskingsprosessen. Dei grensande faktorane for lærarar si utvikling av læringsmåla er mangel på tid og mangel på lærestoff. For å ta høve for dette burde lærestoff som baserer seg på læringsmåla vere tilpassa spesifikke fag og bestå av materiale og aktivitetar som kan brukast direkte i klasseromsundervisinga.

Preface

This is a master's thesis that concludes five incredible years of discovery, learning and development. In the final stretch of finalising this report, I am left with two major results. The first result is the thesis itself. It is an artefact of the academic effort and achievement my educational travel has lead to, and a product I am proud of. The second result is a sense of gratitude. I am grateful to all of the study participants, who have given invaluable insight to the research and fuelled my inspiration to become an educator. I am grateful to my brilliant supervisor Monica for the guidance and the support provided during the entire course of my master year. I am grateful to my friends and family for the constant encouragement and the inspiring conversations. I am grateful to my study programme peer Anne for the the academic and the social motivation to complete this project. The thesis might be the conclusion of my own master's programme, but it most definitely has been a team effort.

1st of June, Trondheim
Viggo Skarby

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In today's society and the technological development of the 21st century, computing is a relatively newfound and central concept. Guzdial (2015) elaborated on the idea of *computing for everyone* and argued for why everyone should learn computing as part of the compulsory education. The term computing for everyone captures a current international trend in the field of education, which manifests in the inclusion of computing in the national curricula of multiple countries in the world. This has happened in countries like England, Wales, Ireland, Spain, France, Finland, Denmark, Sweden and many more (European Schoolnet, 2015, as cited in Sanne et al., 2016, pp. 63-67; Dolonen et al., 2019, p. 16). Reasons for the inclusion of computing in secondary education are future needs for jobs, learning about the 21st century world, computational literacy and broadening of participation in the field (Guzdial, 2015, pp. 101–103). The idea of computing for everyone is also reflected in the educational politics of Norway. In NOU 2015:8 (2015) the future of the Norwegian school was presented. Here, the need for digital competence was highlighted as an essential and ubiquitous part of the need for future competence. The report led to the Norwegian government creating a new National Curriculum for Knowledge Promotion in Primary and Secondary Education and Training (LK20). In the new curriculum computational thinking and digital competence was integrated into various subjects and in the core curriculum (Ministry of Education and Research, 2017).

The constituted working group Sanne et al. (2016) presented a report on how to implement technology and programming in the Norwegian primary and secondary education and training. The recommendation of the working group on how to perform the inclusion of technology and programming in the national education was to introduce a new compulsory subject in primary and secondary education. This subject was to have a specific focus on and curriculum about technology and programming. This proposal was however not followed in the renewal of the Norwegian National Curriculum. Programming was not made into a self-standing compulsory subject, but rather introduced into other subjects like mathematics, natural science, music and arts and crafts. There are subjects in secondary education that address technology and programming solely, but these subjects are electives. The concept of computing for everyone is therefore implemented in the Norwegian primary and secondary education through programming being a part of other obligatory subjects. It is not an explicit subject that is aimed solely at the content of technology and programming.

Computing is the field related to computers, information and technology. It about connecting computer science to other disciplines, such as engineering and social sciences. It is a broad term and includes information science, information technology, software engineering and computer engineering. Programming is a part of computing, and is the process of producing programs and manipulating a computer to fulfil some purpose (Guzdial, 2015, pp. 1–3). Programming is also a broad term and includes more than just the act of writing a computer program. Identifying a problem, designing a solution, writing code for the computer, debugging and continuous improvement of the code are all parts of programming (The Norwegian centre for ICT in education, 2017, p. 9). One of these aspects; debugging, is important for programming. By extension, debugging is an important part of computing. Most professional software developers spend almost half of their time on debugging (Perscheid et al., 2016, pp. 98–99). The great time investment debugging is, displays the importance of debugging as part of programming.

There are many different definitions for what debugging is. Most researchers in the field of computing education agree on that it is a form of troubleshooting and that is entails locating and fixing errors, also called bugs, in software programs (Chmiel & Loui, 2004; Li et al., 2019; Michaeli & Romeike, 2019a; Miljanovic & Bradbury, 2017; Murphy et al., 2010). What classifies as a bug, and what processes fall under debugging, is not unambiguously agreed upon. Defining debugging to be a process after successful compilation and running of a program is prominent, but some also include dealing with compile-time and runtime errors as debugging (Michaeli & Romeike, 2019a, p. 1037).

One might suppose the importance of debugging in programming implies that debugging is a central part of the education of programming. Studies have however shown that in countries like New Zealand and Germany, this is not the case (Li et al., 2019; Michaeli & Romeike, 2019a; Whalley et al., 2021). Debugging is an underrepresented part of the computing education in these countries. In addition, there are no established best practices or pedagogies for educators to follow when teaching debugging (Fitzgerald et al., 2010, p. 390; Michaeli and Romeike, 2019a, p. 1030). This poses a divide between the practice of computing and the education of computing and leaves the teachers and pupils on their own in the development of this essential part of programming and computing in an educational setting.

1.2 Professional relevance

This master thesis is written as a part of the study programme Natural Science with Teacher Education with informatics as main subject at the Norwegian University of Science and Technology (NTNU). The research project and report are relevant for the profession the study programme is connected to. As a final project and report I am conducting a didactic research project and writing a master's thesis. This project and report will be important for my profession in a dual way. The first is the domain of the research project, which is education of computing. The study is set in the context of computing education in Norwegian secondary school, which is the profession the study programme is aimed for. By studying the phenomenon of debugging in Norwegian computing education at secondary level, my content knowledge and didactic knowledge are developed. This will be essential to my imminent profession. In addition, the project is a research study on the teaching situation of a classroom. An important part of the teacher profession is development and progression of the content, pedagogy and didactics of education. This qualitative research study develops my research methodology in social sciences and education, as well as my knowledge on the research field and state-of-the-art of computing education. This will prepare me for the professional small- and big-scale research projects that will be conducted in the professional setting of a teacher.

1.3 Problem definition

Due to debugging being an underrepresented aspect of computing education in other countries, and there being no established best practices or didactics of how to conduct debugging education, it is important to study the phenomenon of debugging education in a Norwegian context. There has been conducted some research abroad on debugging in introductory computing courses at tertiary education level (such as Chmiel & Loui, 2004; Fitzgerald et al., 2010; Miljanovic & Bradbury, 2017; Whalley et al., 2021) and on debugging in K-12 programming and computing subjects (such as Michaeli & Romeike, 2019a, 2019b). These studies mainly find that learning systematic debugging increases debugging efficiency, effectivity and self-efficacy for the learners. Since developers spend almost half of their time on debugging (Perscheid et al., 2017, cited in Michaeli & Romeike, 2019a, p. 1030), debugging is important to address in the education of computing. There is also little research done on debugging education in secondary level and in a Norwegian context. This is therefore an important focus to research.

The existing literature on debugging focuses on the effects of systematic debugging from the learner's perspective. There is however little research done on the teaching and didactics of debugging, and there are no established sets of best practices for debugging strategies nor teaching of debugging (Michaeli and Romeike, 2019a, p. 1030; Fitzgerald et al., 2010, p. 390). To be able to teach systematic debugging in a feasible and learning-oriented way, it is also important to have a focus on the teacher. Since teaching consists of more than merely knowledge of the content to be taught, it is essential for teachers of computing subjects to develop pedagogical content knowledge as well as debugging content knowledge (Shulman, 1986). An important part of teacher learning is knowing and understanding the pupil's perspective and how to guide the pupil towards the learning objectives. Content knowledge and didactics of debugging from a teacher perspective are the foci of this research study.

To look at debugging one needs to decide on a definition, of which there are many. Many researchers in the field agree that debugging is a special form of trouble-shooting in the context of programming, and that it is the process of finding and fixing bugs in a software program (Chmiel & Loui, 2004; Miljanovic & Bradbury, 2017). Most define it as an activity that comes after testing, and only include runtime, semantic, logical and functional errors (such as Chmiel & Loui, 2004; McCauley et al., 2008; Whalley et al., 2021). However, Michaeli and Romeike (2019a) make an argument for that compile-time and syntax errors also should be included in secondary computing education. This is based on the finding that these types of error are prominent in the K-12 computing classroom, and that they pose a major hurdle for pupils. In this research project debugging is defined as the process of detecting, locating and fixing bugs in ones own or others' computer programs. This includes fixing semantic, runtime and functional errors, but also compile-time and syntax errors, seeing as these are a problem area in the secondary computing education classroom.

The aim of this project is to create an overview of what debugging education should entail, set in the context of Norwegian secondary school. The goal is to create an artefact, in the form of learning objectives, for teachers of programming. The learning objectives will be designed to help educators develop their own debugging skills and their knowledge on debugging didactics. The learning objectives should reflect the existing literature on debugging and debugging didactics. They should also be tailored for the context of Norwegian secondary education. The research questions for this project are

> **RQ$_1$**: What do teachers need to know to integrate debugging in their teaching of programming?
> **RQ$_2$**: What conditions and constraints are there for development of teachers' debugging skills?

"Teachers" in the research questions refers to Norwegian teachers of computing subjects in lower and upper secondary school. To be able to integrate debugging in their teaching, one must look at both the content knowledge required to debug and the didactics of teaching debugging. The conditions and constraints are factors that may lay requirements or restrict the possibility for development of the necessary knowledge. To study and discuss the research questions a small-scale qualitative research study based on design science research was designed and conducted. The aim of the study is to design learning objectives that provide a conceptual framework and contextualisation for the professional development of teachers' debugging skills and didactics. The project is split in three iterations: 1) initial definition of learning objectives, 2) expert evaluation and 3) teacher evaluation. The general methodology for the research project is described in the following chapter 1.4. The structure of the report is presented in chapter 1.5.

1.4 Methodology

The research design of this study is placed in the qualitative paradigm. The practice, attitude and needs of persons are individual, and every individual has their own perception of debugging education. An interpretive approach is necessary to study the research questions and the phenomenon of debugging education (Robson & McCartan, 2016, pp. 24–25). A small-scale qualitative study with a flexible design was created. Due to the flexible design of the study, the research method and research questions have been in development and open for change during the entire course of the research project. The study has features of design science research, and the project has gone through different cycles in iterations. See figure 1.1 for the design science research model of the research project. The figure represents the three cycles of design science research as described by Hevner (2007). The cycles are the rigour cycle, the relevance cycle and the design cycle. In this project the rigour cycle is done through a literature review of literature on debugging and debugging education that exists in the knowledge base. The relevance cycle is done through interviews with actors in the environment of debugging education practice. The design cycle is done in three iterations and is based on the environment and the knowledge base. The first step of the design cycle consists of design of learning objectives. The second step the design cycle, evaluation of the learning objectives, was done twice. First with experts on the debugging education field, and secondly with practitioners of debugging teaching in the secondary education classroom.

To discuss and study the research questions and phenomenon in focus, empirical data was collected. The collection of data was done through a literature study, in addition to interviews with educators of computing subjects in different levels of education. The data sources were chosen to give and insight in the current status of debugging education, and to place this in the context of the

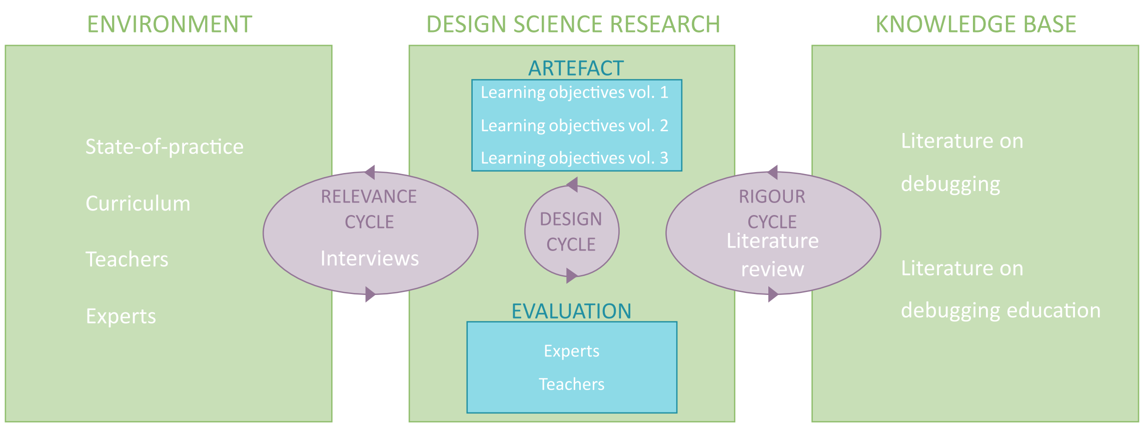

Figure 1.1: The research project placed in design science research

Norwegian secondary school classroom. Since the object of the study is dependent and affected by context, the qualitative research design centres around an in-depth analysis of a smaller data set and performs an analytical generalisation to answer the research questions (Polit & Beck, 2010). The method of data collection and analysis are presented more in-depth for each of the three design cycles. See chapters 4, 5 and 6 for the description and documentation of the three design cycles respectively.

According to Morse (1999, as cited in Robson & McCartan, 2016, p. 169) validity and reliability are essential to make qualitative research rigorous. The validity of the research study focused on awareness of bias in the analysis and results, and on avoiding invalid description. Since the researcher is the most important tool in qualitative research, the analysis and results will inevitably be affected by the experiences and subjective theories of the researcher. The intention of the qualitative analysis conducted in the study was to enter with a mental "tabla raza", a blank slate. During the structuring of the data material it was striven to put the personal perspectives of the researcher aside, and to be aware and avoid deficiencies of the human analyst (Robson and McCartan, 2016, p. 462; Postholm, 2005, p. 86). The reliability of the study was cultivated by providing a thick description. A thick description entails a rich and thorough description of the context, research design and conduction of the study (Polit & Beck, 2010, pp. 1450–1451).

1.5 Outline of report

This thesis reports the conducted research project. Firstly, the environment and knowledge base of debugging education are described in chapter 2 and chapter 3. In the following chapter 4 the 1st design cycle and initial learning objectives are presented. In chapter 5 the 2nd design cycle and expert evaluation is presented. The 3rd and final design cycle is presented in chapter 6, which is the teacher evaluation. Lastly, the implications for design of a learning sequence are discussed and the research questions answered.

Chapter 2

Environment

In this chapter the environment of debugging education is presented. The environment consists of the practitioners in the field and the government regulations that dictate the legalities of debugging education.

2.1 Context

There are many different factors that make up the context of the classroom. Legally there are governing documents and national laws that lay the grounding for the primary and secondary education. Locally at different schools there are regional regulations, school cultures and various resources that enable or limit teaching. Internally, the classroom teaching is guided by the individual teacher and the pupils. This research is placed in the paradigm of sociocultural learning theory, and the interaction between actors in the learning process is essential to and expands the potential for learning and development (Vygotsky, 1978/2018, pp. 156-165; Säljö, 2000, p. 155). While the internal and local environments may vary and change from classroom to classroom, the national regulations lay the foundation for every Norwegian school and classroom. The following description will therefore mostly focus on the general regulations that set the context of every computing classroom in Norway.

The core curriculum of LK20 states the values and principles that the Norwegian education system is built upon. Each teaching subject also has a dedicated curriculum that defines the learning objectives of, motives for and core elements in the subject. In Norwegian secondary school there are two subjects that primarily aim at teaching computing and programming. The first one is an elective in lower secondary school, Programming (PRG01-02), and the other is an elective in upper secondary school, Information technology (IT01-02). Both PRG01-02 and IT01-02 contain different computing aspects. One of the common aspects is debugging, which is shown in the subject curriculum of both electives. In PRG01-02 the core element *software development* is defined as:

> The core element software development is about practical work with planning, developing and continued development of a user-friendly and functional digital product through testing, *debugging* [emphasis added] and adaptation. Furthermore, the core element is about reflection on privacy and sharing culture in software development

> processes (Directorate of Education and Training, 2020, own translation).

Debugging is explicitly mentioned, as emphasised in the quote above, and is therefore an explicit part of PRG01-02. This is also the case for IT01-02, which has debugging explicitly mentioned in the description of basic skills in the subject. The basic skill *digital skills* in IT01-02 is defined as:

> Digital skills in information technology involve using and selecting relevant digital resources to search for, exchange, process and present information. Furthermore, they involve critically assessing sources and showing digital and ethical judgement. Digital skills also involve using relevant and effective tools to develop and *debug* [emphasis added] information systems (Directorate of Education and Training, 2021, own translation).

This description of digital skills in IT01-02 plants debugging as a central part of the curriculum in this subject as well. Therefore, the concept of debugging is central to computing at secondary education level in Norway.

As mentioned in the introduction, studies have found debugging to be an underrepresented part of computing education in other countries. Since research on debugging in Norwegian education is lacking, there is currently little insight into if this is the case of the computing education in Norway as well.

There are no directions for what tools to use in the subject IT01-02. In PRG01-02 the learning objectives dictate that the pupils should learn multiple programming languages, wherein at least one is text-based. Other than this, there are no specifications in the governing documents for what programming language or tools the teaching should revolve around. Even though there are no legal directions for tool use, some norms and best practices have evolved in the teaching community. By looking at different resources for computing subjects, one can get an insight into what the conventions and norms are based on what content is available. For upper secondary school most schools use HTML, CSS, JavaScript, PHP and SQL. In lower secondary school block-based programming languages like Scratch and micro:bit are prevalent, in addition to the text-based programming languages JavaScript and Python (IKT i Stavangerskolen, n.d.; Lær kidsa koding, n.d.; Siljan kodeklubb, n.d.).

Chapter 3

Knowledge base

In this chapter the knowledge base of debugging education is presented. The knowledge base is centred around scientific theories and methods based on empirical research. To create an overview of the knowledge base, a literature review was performed.

3.1 Literature review

In this section the literature review is presented. The review was conducted based on the research questions and the domain of debugging education. In the following the method and the results of the literature review are presented. The results consist of an overview of the read articles, in addition to common themes discovered.

3.1.1 Method

The literature review was conducted in order to create an overview of the knowledge base of debugging education. The contents of the review were existing literature and research on debugging education. The aim of the review was to find out what debugging didactics are being practised in educational settings, and what techniques and strategies research has shown to have educational potential. By understanding the state-of-the-art of the didactics of debugging the knowledge can be built upon and adapted to the Norwegian classroom.

A literature search was done using the ACM database with the search term *[All: debugging] AND [All: education]*. This yielded 10 637 results. To reduce the set of entries, the ordering was set to Relevance and the first 50 entries were considered for inclusion in the literature review. By reading the title and skimming through the abstract, a first set of articles was selected based on apparent relevance to the domain and research questions of the study. The selection was also done with the intention of collecting a broad representation of techniques and processes for debugging education. Initially 10 articles were selected. By snowballing the references 1 additional article was included in the literature review. The total number of reviewed articles was 11.

The articles were read, and common themes, strategies and findings were written down. An overview of the articles in the literature review, the educational levels they focus on and their main contributions are presented in table 3.1. The common themes, strategies and a more in-depth description of the findings are presented in the following section.

3.1.2 Results

Authors (Year)	Education level	Main contribution
Li et al. (2019)	Novice programmers	Framework for debugging.
McCauley et al. (2008)	Novice programmers	Literature review.
Michaeli and Romeike (2019a)	K-12	Identification of relevant skills for debugging. Teachers lack a systematic approach for teaching debugging.
Michaeli and Romeike (2019b)	K-12	Explicit teaching of systematic debugging increase self-efficacy and performance.
Michaeli and Romeike (2020)	High school	Understanding and explanation novice debugger behaviour. Methodology.
Wang and Souders (2012)	High school	Introduction for students to debugging research.
Miljanovic and Bradbury (2017)	Undergraduate	Serious game for learning debugging.
Whalley et al. (2021)	Undergraduate	Novice debugger difficulties. Novice debuggers perception of debugging.
Chmiel and Loui (2004)	Undergraduate	Taxonomy of debugging abilities and habits.
Fitzgerald et al. (2010)	Undergraduate	Student approaches to and difficulties with debugging.
Murphy et al. (2010)	Undergraduate	Discourse patterns for debugger pairs.

Table 3.1: Overview of the contents of the literature review

Researcher and professor emeritus Chmiel and Loui (2004) conducted a study on the effects of formal training in debugging. The study was done in connection to an introductory computing course at a university in the United States of America. The focus of the study was students' development of skills in diagnosing and removing defects from computer programs. The students of the course were given additional debugging tasks. The tasks included extra debugging exercises, debugging logs to log problems and solutions, reflection logs for tracking of skill development and lastly collaborative assignments such as peer code review. The tasks were optional for the completion of the course and therefore split the student group into a test group, the ones who did the optional tasks, and a control group, those who did not. The results of this study are that completion of the optional debugging exercises significantly decreased the time spent on debugging, and that this improvement is not due to aptitude nor program design skills. By building on the Dreyfus model of skill development, the study also provides a model of different stages of debuggers and their respective abilities and habits. The stages are novice, advanced beginner, competent, proficient and expert. Students who completed the optional debugging exercises displayed ability and habits of higher levels of debugger than the control group.

Computing doctorates and teachers of computing subjects at University of Auckland, Li et al. (2019), have adapted the framework of Jonassen and Hung (2006, as cited in Li et al., 2019) for teaching troubleshooting to the debugging domain. Through an intensive literature review on debugging education, they identify and structure debugging processes and knowledge that are fundamental for novice debuggers to learn. The study provides both an overview of the knowledges required for debugging, the steps of the debugging process and debugging difficulties found in the literature on debugging. The knowledge required to debug are 1) domain knowledge, 2) system knowledge, 3) procedural knowledge, 4) strategic knowledge and 5) previous experience. 1) Domain knowledge is knowledge about the underlying programming language, which impacts the ability to debug programs written in this language. 2) System knowledge is knowledge about the program in need of debugging. System knowledge contains both topological knowledge, how the structure of the components of the program is constructed, and functional knowledge, what the functions of a program do and how the components interact. 3) Procedural knowledge is knowledge about how to perform debugging actions. This includes actions such as how to set up a test case, insert breakpoints or inspect memory. 4) Strategic knowledge is knowledge about strategies for effective debugging. The two types of strategic knowledge are global and local strategies. Global strategies are not context-specific and independent of the system. Mentioned global strategies in the paper are discrepancy detection, forward and backward tracing and breath-first searching. Local strategies are context-dependent and specific to a certain program. Meaningful variable names and comments in the program are mentioned as examples. Even though there are two types of strategic knowledge, they are interconnected and may be applied adjoined whilst debugging. 5) Previous knowledge is knowledge about what bug one has encountered in the past and how these bugs were eradicated. Novice and expert debuggers may utilise the same strategy for debugging, but experts form more correct hypotheses than novices and are more efficient at finding the error. In addition to knowledge required to debug, four steps of debugging are presented as an application of the presented knowledge. The first step is *Construct the problem space.* This entails constructing a mental model of the system, such that the debugger may locate the bugs in the program. The second step is *Identify fault symptoms.* By having constructed a mental model of intended and actual behaviour of the program, the debugger may detect discrepancies. The third step is *Diagnose the fault.* By understanding the discrepancy of behaviours, hypotheses of the fault may be made. The hypotheses need to be tested and confirmed to identify what is causing the bug. The fourth and final step of debugging is *Generate and verify solution.* Once the developer knows where and why a bug occurs, they can generate and test solutions until the bug is fixed. It is important to also check for implementation of new bugs in the program created by the solution. Finally, the report identifies debugging difficulties for both novices and experts. Novices primarily have difficulties constructing the problem space. Program chunking is one technique that separates the novices and experts in this regard. Time spent on identifying the problem before trying solutions was also a main difference. Experts spend more time identifying the problem before trying solutions than novices do, which causes novices to spend more time doing debugging that does not advance a solution. Lastly, considering alternatives is identified as a difference between novices and experts. Whilst experts consider multiple possible solutions, and discards unsuccessful hypotheses, novices often use a depth-firth approach to debugging and are less likely to discard an unsuccessful or unsatisfactory hypothesis. All these habits highlight what difficulties novices have with debugging compared to experts.

Professors of computer science at University of Ontario Institute of Technology, Miljanovic and Bradbury (2017), designed a serious game designed to help students learn effective debugging techniques and to make this learning more enjoyable and motivating. The game RoboBUG is aimed at first-year computer science students and is created to both learn the students the material and to demonstrate critical thinking and problem-solving. The different levels of the game introduce and train the player in different debugging techniques in C++. The techniques introduced in the game are code tracing, print statements, divide-and-conquer and breakpoints. By doing a pre-post-test they evaluate the enjoyability of the game and the improvement of the students' understanding of debugging techniques. The study concludes that the game helps students improve their understanding of debugging techniques, especially for students who initially were complete beginners of debugging. However, the game did not significantly impact the enjoyment of learning.

McCauley et al. (2008) have performed a literature review on debugging research from an educational perspective. The main contributions of their study are why bugs occur, what types of bugs occur, what the debugging process is, and how this knowledge can improve teaching and learning of debugging. The question of why bugs occur is concluded to not have a simple answer, but that it boils down to a cognitive breakdown in the programmer. The root of the cognitive breakdown is either misconceptions, fragile knowledge or the term "superbug" introduced by Pea (1986, as cited by McCauley et al., 2008, pp. 70-71; as cited by Guzdial, 2015, p. 30). The superbug refers to the strategy that the computer has a mind of its own with interpretive powers. For novice programmers it appears to be some errors that occur independent of programming language or paradigm. These are off-by-one errors, operator precedence errors and misplacement of code in a loop. Some commonalities the authors found in the literature was that debugging consists of discovering bugs by checking known values, gaining topological and functional knowledge of the program in question and repairing the error. Novices often spend less time on creating a hypothesis for debugging and often get stuck in a depth-first approach. For novices it is also a common problem that their attempts at fixing a bug creates new bugs. The implications of the literature review for teaching and learning debugging are a set of improvement suggestions made by the authors. Their recommendations are that debugging education needs to combat preconceptions, misconceptions and fragile knowledge, that it should build program comprehension skills, that debugging skills should be explicitly taught and that tools used need to not interrupt the student's ability to comprehend code and work productively.

Whalley et al. (2021) conducted a study on the reflections of novice debuggers. The aim of the study was to see if learning a structured, formal process for debugging would be seen as valuable to the students. As part of their discussion, the authors present some implications of their study for the teaching of debugging. One of the focus areas is attitudes towards debugging and the teacher understanding of how to manage student frustration around debugging. Reflection through collaborative activities such as pair debugging is suggested as a viable approach to deal with attitudes and frustration. A systematic process for debugging is also seen as fruitful for students, especially for larger and more complex debugging endeavours.

A central pair of researchers in the field of debugging education, Michaeli and Romeike (2019a), performed a study to investigate how debugging is conveyed and taught in the K-12 classroom. They found the general debugging processes to share some aspects, such as testing, gaining overview, formulating a hypothesis, verifying the hypothesis, refining the hypothesis, correcting the error and re-testing the program. They present a total of 18 findings, regarding how and why debugging is taught (or not taught). The main contribution is that students and teachers lack a systematic process for debugging, which leads to ineffective teaching situations and frustration with debugging for both pupils and teachers. As a help to create concepts and material for the classroom, computing education should according to the authors be primarily targeted towards "weak" to "average" students' requirements, focus on self-reliance and supporting it, emphasise a high-level systematic debugging process, include approaches for coping with compile-time errors and introduce debugging strategies and tools systematically.

Michaeli and Romeike (2019b) also conducted an intervention study to analyse the effectiveness of explicitly teaching a systematic debugging process in the K-12 classroom. They found the teaching of a systematic debugging process to have a positive impact on students' self-efficacy and performance in debugging. To investigate students' pre-existing debugging traits, Michaeli and Romeike (2020) performed a cross-sectional study where students solve an escape room. The findings of this study also report that students struggle to form hypotheses and approaching a problem breadth-first. Other prerequisites student had, were struggle with decomposition of problems, struggle with reversion of attempts at fixing bugs, struggle with gaining topological knowledge of a system and that students easily become frustrated with the troubleshooting process.

All of the research studies in the literature review focus on a set of debugging strategies and conceptual knowledge for debugging. While the pedagogical approaches in the articles vary, the debugging strategies and conceptual frameworks for debugging are similar and repeated through them. While there may be some differences between the articles in definition and implementation of the debugging strategies, there are five main strategies that are recurring. The five strategies identified in the literature are 1) program tracing, 2) discrepancy detection, 3) program chunking, 4) bug prevention and 5) collaborative development.

Li et al. (2019, p. 81) report that tracing is a commonly used strategy, and that both forward and backward tracing are beneficial and useful strategies to learn for debugging. Miljanovic and Bradbury (2017, p. 94) also use code tracing as one of their four debugging techniques, and place the strategy as one of the fundamental techniques for debugging. The findings of Michaeli and Romeike (2020) show that even though pupils utilise debugging strategies, they struggle with the cognitive load of tracing. Program tracing is reported as a fundamental and commonly used debugging strategy, but something pupils struggle with. Especially a focus on the ability to backwards trace is mentioned to be beneficial in the literature.

The second general debugging strategy present in the literature is program chunking. Program chunking is one of the other techniques utilised by Miljanovic and Bradbury (2017, p. 94), where it is called divide-and-conquer. Li et al. (2019, p. 83) bring forth program chunking as a debugging difficulty for novice debuggers, and show that poor program chunking abilities is an indicator of poor debugging skills. According to the research of Spinellis (2018, cited in Michaeli & Romeike, 2019b, p. 2), the strategy provides the developer with information to help with localisation of the bug. The strategy is here called slicing, but it entails splitting the program into chunks of code to pinpoint the faulty code. Program chunking is an effective strategy for locating bugs in computer programs.

The third general debugging strategy is discrepancy detection. Li et al. (2019, p. 81) highlight how discrepancy detection can be both a global and a local strategy. Detection of discrepancies is often done through printing of system state and is a commonly used strategy for locating bugs. Another method for discrepancy detection is the use of a debugger tool, such as displayed by Miljanovic and Bradbury (2017) and discussed by Chmiel and Loui (2004). The debugger is commonly used to aid the debugging process, but the complexity and function of the tool depends on the integrated development environment. This makes discrepancy detection a strategy that can require procedural knowledge. Since this local strategy also requires the developer to know the expected output, system knowledge is another central need for discrepancy detection.

The fourth general debugging strategy is collaborative development. This may be in the form of collective reflection, such as described by Michaeli and Romeike (2019b), or methods like pair programming. Pair programming is a collaborative development strategy in where one person is the *driver* and another is the *navigator*. The navigator tells the driver what to write, and the driver implements the oral instructions from the navigator into program code (Agile Alliance, n.d.). Pair programming is proposed by McCauley et al. (2008, p. 85) as means to help prevent faulty preconceptions, misconceptions and the fragile knowledge of pupils, which hinder successful debugging. Murphy et al. (2010) also present a collaborative activity, pair debugging. Pair debugging, like pair programming, is a process where one person is the driver who manipulates the code, while another is the navigator who decides what the driver should do. Pair debugging seems a promising technique for bringing out the though process of the students. From a pedagogical point of view, it is also promising in the way of making students better programmers and collaborators. The technique helps the pupils convey their domain knowledge, system knowledge, procedural knowledge, strategic knowledge and their previous experiences to the teacher and to their fellow pupils. Pair debugging is also discussed by Whalley et al. (2021, p. 78) as means to facilitate reflection and meta-cognition. A final collaborative activity for debugging found in the literature, used by Chmiel and Loui (2004, p. 18), is peer code review. This debugging strategy makes the detection and locating of bugs a collaborative effort.

The fifth general debugging strategy is bug prevention. Naming conventions and commenting of code, highlighted as strategic knowledge by Li et al. (2019, pp. 81–82), are two strategies for the prevention of bugs. In their framework for debugging, system knowledge is a requirement for debugging. Developing the system knowledge to understand the functionality and topology of a program is also a way of improving the debugging process and preventing errors in the programming process. Planning and creating a mental model of the program in advance are means for preventing bugs. Also, the development of procedural knowledge on how to use the programming language and the available tools to perform debugging can be ways of preventing bugs.

Another common point in the literature is the importance of verifying the solution. This may be in the form of checking if the hypothesis and implementation fix the bug, and reverting unsuccessful attempts and trying again. Another type of verification is checking if a fix of a bug leads to bugs elsewhere in the program. Reversion is an important part of the systematic approach to debugging presented by Michaeli and Romeike (2019b, p. 5), and is part of all three steps of their approach. Fitzgerald et al. (2010, p. 392) also highlight the use of the "Undo"-button in the development environment as a debugging technique.

One of the central findings in the literature is that a systematic debugging approach can facilitate the improvement of debugging skills. The debugging process is dependent on previous knowledge and experience about bugs and strategies for finding and fixing bugs. Doing debugging exercises and activities are proposed as beneficial for student development and improvement of debugging skills. This is part of the findings and recommendations of Li et al. (2019) and Michaeli and Romeike (2019a, 2019b, 2020). The literature also recommends the teaching of debugging to be explicit (Chmiel & Loui, 2004; Fitzgerald et al., 2010; McCauley et al., 2008).

A common aspect in the definitions of debugging is that it is a problem-solving process. Also, Michaeli and Romeike (2017) claim that debugging in a broad sense is a computational thinking approach. The specification of what types of bugs one solves during the debugging process is mostly defined in the articles. Most of the literature defines compile-time error handling, or syntactic and some semantic errors, as not a part of debugging. The literature is also mostly united on that debugging is the problem-solving process of runtime, or semantic and logical, errors. This specifies that fixing of bugs that are located by the compiler is not a part of the debugging process. The exception is Michaeli and Romeike (2019a) which propose that approaches for coping with compile-time errors should be a part of debugging teaching due to the high percentage of these types of error in the K-12 classroom. The literature also acknowledges the importance of self-reliance to avoid “learned helplessness”, which is the opposite of self-efficacy. The findings of McCauley et al. (2008) and Michaeli and Romeike (2019a) focus on the explicit teaching of debugging strategies and development of self-reliance as important implications for the teaching of debugging.

A breadth-first approach is an important difficulty novice debuggers struggle with. Fitzgerald et al. (2010) found considering alternatives to be a strategy students at university level introductory courses in programming struggle with. This is also a difficulty highlighted by McCauley et al. (2008) and Michaeli and Romeike (2019a) and is an important part of the framework by Li et al. (2019).

Chapter 4

1st design cycle: Defining learning objectives

The 1st design cycle consisted of design of the initial learning objectives. The learning objectives were designed for teachers of lower and upper secondary school who teach computing subjects. The aim of the learning objectives is to contextualise and specify what debugging knowledge and skills these teachers need to develop to integrate debugging as a part of their teaching. The design of the initial learning objectives was based on the environment and the knowledge base. The environment provided limitations, opportunities, needs and context. The knowledge base provided best practices, frameworks and processes for debugging that are founded in research.

4.1 Method

The creation of the learning objects was done in iterations of the design cycle. The initial set of learning objectives was created heavily based upon the literature review, which provided an overview of the knowledge base. In addition to the knowledge base, the learning objectives were designed with regards to the governing documents that define some of the context in the environment of debugging education. During the first iteration of the design cycle 5 learning objectives were created. The learning objectives are presented in table 4.1. In the following section the grounding for each learning objective is specified.

4.2 Initial learning objectives

Learning objective A_1 entails knowledge on different types of bugs. As concluded by Michaeli and Romeike (2019a, p. 1031) there are various categorisations for different types of bugs. The debugging process will vary depending on the underlying type of error (Michaeli & Romeike, 2019b, p. 2). The presented types of categorisations are syntactic, semantic and logical errors, construct and non-construct related errors or compile-time and runtime errors. These categorisations of bugs are displayed in the literature. The systematic framework for debugging presented by Michaeli and Romeike (2019b) distinguishes between compile-time, runtime and logic errors. Chmiel and Loui

ID	**Learning objective**
A_1	The teacher should have knowledge about different types of bugs
B_1	The teacher should know how to use some global and local debugging strategies
C_1	The teacher should have an understanding of why debugging is an important part of computing education
D_1	The teacher should know some techniques for promoting problem-solving and self-reliance in the pupils
E_1	The teacher should know how to revert unsuccessful attempts at fixing bugs

Table 4.1: Learning objectives after the 1st design cycle

(2004) on the other hand use the categorisation of syntax, semantic and logic bugs. Both articles specify that different types of errors require various and different strategies and techniques to be fixed. Due to this, it will be a necessary foundation for teachers to have knowledge on different types of bugs and what approaches each of them requires.

Learning objective B_1 entails knowledge on debugging strategies. Li et al. (2019) give a description of what local and global debugging strategies are, and that this strategic knowledge is an important factor in being able to debug. Therefore, it is important that teachers develop their own strategic knowledge by learning some debugging strategies. They also need to know how to implement these in the classroom. There are multiple debugging strategies presented in the literature on debugging education. While the name and details about a specific strategy may vary, the strategies in the articles from the literature review can be summarised in five general strategies. The five general strategies are program tracing, program chunking, discrepancy detection, collaborative development and bug prevention. All of the five general debugging strategies are important parts of learning objective B_1.

Learning objective C_1 entails the importance of debugging as part of computing subjects. Guzdial (2015) focuses on the concept of computing for everyone, which is reflected by the new National Curriculum in the Norwegian school system. The National Curriculum integrates computing in the core curriculum, compulsory subjects and electives. It is also reported that debugging is a big and time consuming part of computing (Perscheid et al., 2016, pp. 98–99), but that debugging is missing from computing education (Li et al., 2019; Michaeli & Romeike, 2019a; Whalley et al., 2021). According to the debugging framework of Li et al. (2019) there are multiple knowledges required to debug. In order to fill the gap in computing education and develop the required knowledges, motivation and contextualisation of the concept debugging in the computing classroom will be important. This with the aim of helping the teacher to implement debugging as part of their teaching. The focus should be on both the content knowledge of debugging, but also the pedagogical content knowledge of how to teach debugging. Parts of this will be common mistakes, such as the seven causes of bugs described by Spohrer and Soloway (1968, cited in McCauley et al., 2008, p. 70), and difficulties for novice debuggers as described by Li et al. (2019, p. 83).

Learning objective D_1 entails promotion of pupils' problem-solving and self-reliance. The learning objective was designed to address the concept of learned helplessness, which Michaeli and Romeike (2019b) made an approach to avoid. Their approach aimed at fostering self-reliance and tackle the trial-and-error approach to debugging. Both learned helplessness and the trail-and-error approach to debugging should be unlearned to improve pupils' debugging skills. Their teaching of a systematic debugging process included questions such as "Is the program compiling successfully?", "Does the program run without errors?" and "Do the expected and actual behaviour match?". This to focus on compile-time, runtime and logical errors. The questions "What is the cause?" and "Why is the cause?" were also part of the approach to prompt self-reliance and reflection. Whalley et al. (2021, p. 74) incorporated questions to prompt a step-by-step approach to detecting and locating a bug, and creation of a hypothesis for solving the problem. These questions and approaches are important means to develop learning objective D_1.

Learning objective E_1 entails reversion of unsuccessful attempts at fixing bugs. As highlighted by both Michaeli and Romeike (2019b, p. 4) and McCauley et al. (2008, p. 81), it is common for novice debuggers to try multiple fixes at once until the bug is fixed. This may introduce new bugs, create unstructured code and make the developer frustrated and give up. Reversion of unsuccessful bugs is therefore an important step to progress from a novice debugger to an expert one. Reversion of unsuccessful attempts at fixing bugs is an important part of the systematic debugging process of Michaeli and Romeike (2019b), and they include it in all three steps of their approach. The importance of this is manifested in learning objective E_1.

Chapter 5

2^{nd} design cycle: Expert evaluation

The aim of the 2^{nd} design cycle was to evaluate the initial set of learning objectives and bring them closer to the context of the Norwegian computing classroom. An expert evaluation was done with professors of introductory programming subjects at university level. In this chapter the method and results of the expert evaluation are presented, as well as the implications this had for the next iteration of learning objectives.

5.1 Method

To evaluate and contextualise the learning objectives for teachers of computing in secondary school, an expert evaluation was conducted. The evaluation was based on individual interviews with professors and lecturers at university level. They are experts on debugging education in a Norwegian and global setting due to their background in teaching introductory programming courses. In the following sections the method of data collection and data analysis are presented, as well as the ethical aspects.

5.1.1 Interview

The interviews were designed and conducted after the description of Robson and McCartan (2016, pp. 285–307). Due to the flexible research design and the aim of studying experiences, semi-structured focused interviews were conducted. The interviews started with a casual conversation where the aim of the research project was presented. This to give the interview object the opportunity to ask questions before the formal part of the interview. Afterwards the recorded interview began. The interviews were based on interview guides created in advance. The guides contained order of events plus general themes and questions to be focused on during the interviews. Since the interviews were semi-structured the questions were open to adapt based on the answers of the interview object. However, the questions were contained within the domain of the general themes debugging and debugging education. See appendix A for the interview guide containing the structure of the expert interviews. The interviews were recorded with an external recorder, and the recordings were transcribed. All of the interviews were performed physically, and they were conducted in Norwegian.

5.1.2 Selection

The selection of experts for the evaluation of the 2nd design cycle was done strategically. To be considered an expert on debugging education in a Norwegian context, lecturers of introductory programming at university level were invited to be interviewed. The aim was to include multiple aspects and contexts of programming education. Lecturers from three different disciplines were selected: computer science engineer studies, teacher education studies and programmes for in-service teachers. All of the selected experts taught subjects of programming to the different groups of learners at an introductory level. See table 5.1 for an overview of the experts and the courses they taught. Due to the common domain of programming and debugging, some of the questions were similar in the interviews. However, due to the different contexts and target learners of the experts some of the questions differed from interview to interview. This with the intention to focus on their specific learner group and context. The specific questions aimed at exploring the specialisation of the professional field and experience of the expert. The foci of the questions were debugging and the teaching of debugging. See appendix A for the interview guides used in the expert interviews. There were three different sets of questions, one for each expert's domain of teaching. During the interviews the current iteration of learning objectives was presented, evaluated and discussed in light of the expert's teaching domain and experiences. This was done last in the interviews to primarily keep focus on their own practices and experiences without leading the responses of the interviews.

ID	**Teaching subjects**
α	Procedural and object-oriented programming
	Object-oriented programming with Python, C++ and Java
β	Teacher education: Teaching informatics
	Mathematical modelling and ICT in the teaching of mathematics (for grades 8-10)
γ	Introductory programming for teachers
	Applied programming for teachers
	Basic programming with Python

Table 5.1: Interviewed experts and their subjects

5.1.3 Analysis

The manual transcriptions of the expert interviews were analysed. Firstly, the material was coded using constant comparative method as described by Robson and McCartan (2016, pp. 467–481). Common themes were found in the codes and structured into a thematic network. This network was then interpreted to evaluate the learning objectives and design the next iteration. To code and analyse the data material the software program NVivo was used.

5.1.4 Ethical aspects

In research it is important to conduct the study in an ethical way. This is especially significant when involving people, and both their personal and professional integrity must be respected and maintained. In this research study measures were taken to ensure the protection of privacy and dignity of the participants. To ensure the protection of the informants, the study was reported to and assessed by the Norwegian centre for research data (NSD). The assessment and approval from NSD is included in appendix E. To further ensure an ethical approach for the study the guidelines of NTNU (n.d.) for collection of personal data for research projects were followed.

In order to maintain the integrity of the participants and perform data collection in an ethical way, the interviewees had to give informed consent to the participation in the research study. To achieve this, the participants had to sign a consent form which gave consent to collection and processing of data. The consent form endeavoured to fully explain the aim of the research project, what participation would mean for the informant and how the data would be stored, processed and deleted. In addition, the intent and measures taken to maintain the privacy and integrity of the informant were given on the form. The consent form was given to the informant upon invitation to the research study. This with the aim to allow them to read and understand the conditions of participation in advance and allow the participant to give a truly informed consent. This also opened for eventual questions about the research study to be asked and answered before the collection of data began. See appendix C for the consent form for the experts. To build the ethos and ground the selection of the participants, some background information about the experts is provided. The experts were given the option to give consent to be completely anonymised in the final report, or to allow for their teaching position and subject to be included. All of the experts gave their consent to be presented by name and position. Only the teaching subjects are disclaimed in this report.

5.2 Results

One of the main themes discussed in the expert evaluation was how debugging is a part of the expert's teaching. In the teachings of the experts there might occur some explicit teaching of debugging, but it is mostly a supplementary part of the other teaching. Expert α shared how testing is a debugging aspect taught explicitly in their course. When asked if this is how debugging is part of their teaching, expert α answered

> We did not look at [testing] until lecture number 14, how one can have a more structured way of testing. But during the entire course we have compiled our code in debug mode.

Some aspects of debugging, like testing in this case, are taught explicitly. However, debugging is part of the teaching of other computing concepts as well. All of the experts reported to using debugging during teaching as a pedagogical tool. Prominent was the use of live debugging in the integrated development environment's debug mode. The pedagogical goal behind live debugging expressed by the experts was to slow down the execution of code, in addition to display how the debugger tool functions.

Another common theme in the expert evaluation was the usage of tools. When talking about a developer environment used in secondary school called Spyder, expert α said

> And then it is like, the most important thing is to know how to use the tool. Effectively. And I do not know if [Spyder] supports.. Is there a debug mode for example? Is it possible to run code line-by-line? That is at least something I have learnt much from, just seeing what the machine is doing.

How to use the programming tools and integrated development environments were skills the experts highlighted as important parts of learning to debug. Especially the use of debug mode was focused on as a technique the experts used in their teaching, and which made the debugging process easier and more effective.

One common trouble students have, according to the experts, is where to start the debugging process. When asked about the difference between students who are novice and advanced debuggers, expert β said

> I am under the impression that everyone is a bit different. Some like to sit on their own and and tinker with [the program], and I have always been like that. I will figure the problem out by working on it and debugging. While others give up at once. I think there are some differences there.

A common problem for many students is where to start the debugging process. Experts α also pointed out that a good answer to questions like this is to refer them to the use of the debugger tool to locate the error.

An important theme from the expert evaluation was the creation of a culture for debugging, and an expectation that debugging is an essential and time-consuming part of programming. When talking about difficulties with teaching debugging, expert γ said

> And it is important that teachers also are prepared, and that they create.. try to create a culture with the pupils that sets an expectation.. that they factor in time for that it might take 5 minutes to write the code, but in reality it can take 2 hours to make it work.

According to the expert, building a classroom culture and an expectation for spending time and effort on debugging is an important part of programming teaching. The expert also proposed some techniques for creating this classroom culture for debugging. Two of the proposed techniques were live debugging and collaborative activities with a focus on debugging.

5.3 Implications

The 2$^{\text{nd}}$ iteration of learning objectives are presented in table 5.2. The updates in the learning objectives were based on the results from the expert evaluation. The implications of the results are described in the following paragraphs.

The experts reported to mostly using debugging as a support to learning other computing concepts. This is also reported in other research (Michaeli & Romeike, 2019a). However, the recommendation of the knowledge base is that debugging should be taught explicitly in computing education (Fitzgerald et al., 2010, p. 395; Michaeli and Romeike, 2020, p. 9; Michaeli and Romeike, 2019b,

ID	**Learning objective**
A_2	The teacher should know of different types of bugs
B_2	The teacher should know how to use debugging strategies
C_2	The teacher should know how to promote and nurture a classroom culture for debugging
D_2	The teacher should know how to promote problem-solving and self-reliance in the pupils' debugging
E_2	The teacher should know how digital tools can aid debugging

Table 5.2: Learning objectives after the 2nd design cycle

p. 6; McCauley et al., 2008, pp. 86–87). This again strengthens the motivation for learning objective D_2 about promoting problem-solving and self-reliance. Explicitly teaching a systematic process for debugging is not something that is reported in the expert evaluation. Practical debugging strategies for a systematic debugging process are important to address this divide between the state-of-practice and the recommendation of the knowledge base.

To develop a classroom culture for debugging both live debugging and collaborative activities are recommended by the experts. As found in the expert evaluation, the creation of a culture for debugging should be an important part of computing education. The culture should create the expectation of debugging being a time-consuming and essential part of programming. This may also relieve frustration in the pupils when they (inevitably) encounter errors in their own programs or the programs of others. Reflection on learning and debugging helps learners to both learn from errors and to counteract negative emotions and reactions to the debugging process (Whalley et al., 2021, p. 78). Collaborative debugging strategies may also create a culture for debugging with others in the classroom, and live coding or live debugging may highlight the reality of debugging in real coding examples. Due to the importance of creating a classroom culture for debugging, learning objective C_2 was added. This learning objective entails the promotion and nurturing of a classroom culture for debugging as described, and replaces learning objective C_1 from the 1st design cycle. Techniques to achieve learning objective C_2 can be live debugging, collaborative activities and reflection on debugging.

The result that students often struggle with where to start is in accordance with the knowledge base. As mentioned above, the explicit teaching of a systematic debugging approach is recommended to improve the teaching and learning of debugging (Fitzgerald et al., 2010, p. 395; Michaeli and Romeike, 2020, p. 9; Michaeli and Romeike, 2019b, p. 6; McCauley et al., 2008, pp. 86–87). One of the main difficulties students have with regards to debugging is revolved around the creation, modification and revising of hypotheses (Li et al., 2019). By providing a systematic process for debugging, the starting point of the troubleshooting process is specified and defined. A step-by-step approach to debugging and a focus on breadth-first, like presented by Li et al. (2019), could help the students in the creation of a hypothesis and provide a defined starting place for the debugging process. The teaching of specific debugging strategies is also a way of giving the pupils tools and techniques that may aid them in making, re-evaluate and discard hypotheses.

In the expert evaluation, learning objective E_1 was only briefly discussed. Most of the discussion on this learning objective revolved around either debugging strategies, like test-driven development and version handling, or usage of tools, such as undo functionality and debugger mode in the integrated development environment. Since these are either debugging strategies or connected to usage of tools, learning objective E_1 falls either under learning objective B_2 or in relation to tools. Learning objective E_1 was therefore removed from the set, and a new objective about tool use was added. The new learning objective is described in the following paragraph.

Knowledge on the usage of tools is an aspect of debugging education the experts highlighted in the evaluation. Both as a debugging skill, in that one knows how to effectively find and fix bugs, but also as a part of teaching debugging. To help alleviate pupils' frustration in the debugging process, teachers should learn pupils how to use tools to fix bugs and visualise (Fitzgerald et al., 2010, pp. 395–396). Also, compile-time errors pose a major hurdle for pupils (Michaeli & Romeike, 2019a, p. 1034) and learning how to understand and fix these types of error could help pupils avoid frustration in fixing them. Developing domain knowledge and procedural knowledge of how programming languages work can be achieved by searching online. Learning from similar problems and reading documentation will be important skills for pupils to develop in order to improve their debugging. In extension, teachers need to develop the same skill, and learn about problems pupils often struggle with. In the expert evaluation, expert α noted that explanation of compiler error messages is something that should be in the computing education classroom. They also enlightened how knowledge on use of tools like the debugger has been a meaningful learning activity for the students and for themselves. As a result, learning objective E_2 in table 5.2 was added, which entails knowing how digital tools can aid the debugging process.

Chapter 6

3^{rd} design cycle: Teacher evaluation

The 3^{rd} design cycle aimed at evaluating the second iterations of learning objectives and bringing them into the context of the computing classroom in secondary school. Interviews with teachers of computing subjects in lower and upper secondary school were conducted. In this chapter the method and results of the interviews are presented, as well as the implications this has for the final iteration of learning objectives.

6.1 Method

To evaluate the learning objectives for educators and place them into the context of the computing classroom in secondary education level, interviews with teachers were conducted. This was done to provide a final evaluation of the learning objectives by the target group. The interviews were conducted individually with teachers in lower and upper secondary school that teach programming subjects. Since the learning objectives were aimed at teachers of secondary schools that teach computing subjects, the interviewed teachers are a part of the target group. In the following sections the description of the method and results of the interviews are presented.

6.1.1 Interview

Similar to the interviews done in the 2^{nd} design cycle, the interviews in the 3^{rd} design cycle were designed and conducted after the description of Robson and McCartan (2016, pp. 285-307). The interviews were semi-structured and had focus on teaching of debugging, evaluation and relevance of the learning objectives. Before the recorded interview began, a relaxed conversation was initiated to present the project and allow for questions to be asked in advance of the recorded session. Since the interviews were semi-structured they followed the same general lines, but the specific questions asked in each interview varied based on the responses of the interviewee. An interview guide was created and laid the general direction of the interviews. See appendix B for the interview guide used in the interviews with the teachers. The teacher interviews were recorded with an external recorder and later transcribed to allow for analysis of the data material. The interviews were conducted in Norwegian, some physically and some digitally. In the interviews the learning objectives from the 2nd design cycle were presented first. This was with the aim of discussing the relevance, challenges and opportunities of the learning objectives. The foci of the interviews were to evaluate the learning objectives from a teacher perspective, and discuss and uncover implications for design of a learning sequence based on the learning objectives.

6.1.2 Selection

The second round of interviews was with the target group of practitioners in the classroom, which are teachers of computing subjects at lower and upper secondary schools. The selection of teachers was therefore done on the basis of level of education taught at and the relevant teaching subjects. The aim of the research study is to develop learning objectives for teachers of computing subjects in secondary school, and teachers in both lower and upper secondary school that teach a computing subject were invited to participate in the study. See table 6.1 for an overview of the interviewed teachers, the level of education they teach at and the computing subject they teach.

ID	Level of school	Subjects
δ	Upper secondary school	Information technology
ϵ	Upper secondary school	Information technology 1
ζ	Lower secondary school	Programming
η	Upper secondary school	Information technology and media production
θ	Lower secondary school	Programming

Table 6.1: Interviewed teachers and their subjects

6.1.3 Analysis

The analysis of the teacher interviews was done in the same manner as the analysis of the expert interviews, which is described in detail in chapter 5.1.3. The analysis was done on the transcriptions of the interviews using constant comparative method (Robson & McCartan, 2016, pp. 467-481). After common themes in the data material were found and structured, the analysis was interpreted into implications for the learning objects and implementation of the learning objective.

6.1.4 Ethical aspects

The ethical concerns and measures taken to respect and maintain the privacy and integrity of the research project participants were the same for this design cycle as for the last. See chapter 5.1.4 for an in-depth description of the measures taken to perform the research study in an ethical manner. The assessment and approval from NSD regarding the data collection and processing connected to the 3rd design cycle is included in appendix F. The interviewees still had to give an informed consent to the collection and processing of data. The difference between the consent forms for the expert and the teacher interviews was that the teachers did not have to option to give consent to their name and title being in the final report of the study project. All the teachers are anonymised in the processing and presentation of the data. See appendix D for the consent form for the teachers.

6.2 Results

The teachers reported to using debugging as part of their teaching in computing, but mostly in an implicit way. Teacher θ explained how debugging is a part of their teaching like this:

> And I try to [teach debugging] as an integrated part of the other things we do. And to show common errors when I help the pupils individually. In a way that it is not separate, I do not have a separate teaching plan for debugging.

All of the teachers expressed that the teaching of debugging is something that happens with pupils individually or in smaller groups. The teaching is mostly prompted by pupils encountering bugs and requesting assistance. Some teachers also reported to conducting full class sequences on bugs many students met, but these sequences were also prompted by pupil needs in an unplanned manner.

A common reported factor in the difference between pupils was the degree of self-reliance in the debugging process. Teacher δ put it like this:

> Debugging is one of the things that separates the high achieving pupils from the rest. The ability to debug one's own code, and to solve problems. That is what really separates the ones who manage and the ones who struggle. Not necessarily understanding code and programming things, they know that. They understand if, else, for, while and all of that, but when they get stuck they just give up because they do not have this built-in. It is a very big difference.

The pupils who are high achieving in the computing course are also self-reliant in the debugging process. The pupils who do not initiate debugging on their own accord are also the pupils who struggle with other programming aspects. This is reported by multiple of the interviewed teachers. Pupils who do not start the debugging process on their own, or rely on teacher supervision to perform debugging, are a challenge for teachers in the computing classroom. The teachers also reported that pupils usually develop their debugging skills during the course, and fewer pupils will ask for help before they have tried on their own towards the end of the computing courses. One of the main hurdles reported is that pupils lack self-reliance and perform learned helplessness in their debugging.

An important part of becoming self-reliant in the debugging process is the acquisition of knowledge on what the bug is and how others have solved similar problems. Teacher η brought up the importance of knowing how to search online (in the following called *googling*). This was the response as to how this is a part of their teaching:

> I often use it if a pupil has a problem. Especially if they get an error code, or get an error message, but they do not know what it means, I will ask: Have you tried to google it? And then they will be like: No, I have not. Because when you start working in the industry you will spend time on googling things, and you will work with many different frameworks in the private sector. There is not really any reason to focus on neither language or framework, because that world has grown too big. So it is more about focusing on understanding, and the strategies you mentioned, that are important to learn. And the ability to teach one self.

Finding similar problems, and the solutions to them, is an important part of this teacher's teaching. Understanding of computing concepts, debugging strategies and the ability to acquire new knowledge is drawn forth as essential learning for the pupils.

The one constraint all of the interviewed teachers mentioned is the lack of time. Teacher η expressed it like this:

> The main constraint for teachers to develop [debugging competence] is time. (...) Computing is a field that develops rapidly. I think teachers that teach computing should have less teaching hours, so that they can update and develop on the field. For example, in JavaScript new frameworks are released all the time. We mostly work project oriented, and the pupils can choose project and framework on their own. It is not possible as a teacher to know them all.

The field of computing if a rapidly developing one, and there is a need for continuous professional development for teachers. In addition, much of the computing education is based on pupil projects and give opportunity for the pupils to choose frameworks and tools on their own. Knowing every framework and tool is an overwhelming task, and keeping updated on the field of computing is also a continuous struggle against the clock. To focus on debugging, in addition to all of the other aspects of computing, is therefore a difficult task to fit into the allotted hours.

One of the major constraints for the development of debugging knowledge and skills expressed by the teachers was the lack of learning and teaching material. When talking about the role debugging has in their teaching, teacher δ said:

> [Debugging] is one of the things one should focus on, but it takes time to create teaching plans. Because it is difficult to go from debugging to.. This is a classic example of a domain I could use a textbook. Because then someone has taken debugging as a concept and split it into smaller parts, and created an order of teaching and a conceptual mapping of the terms. (. . .) It would be brilliant to have some good, pedagogical snippets of code for the pupils to debug.

There is an expressed desire from most of the teachers to develop their debugging skills and knowledge, especially in a pedagogical context. As mentioned above, time is a prominent constraint for teachers' professional development. The reduction of the concept debugging to a set of learning objectives for the pupils, and the updating on state-of-the-art for both debugging and debugging education, is out of scope for most teachers in the short run. The expressed desire is for a set of learning objectives and pedagogical activities for debugging in the computing classroom.

Another constraint to the professional development of teachers' debugging skills and knowledge is motivation. When talking about the time constraint related to teacher debugging development, teacher ζ said the following.

> And [the time constraint] also leads to some teachers getting demotivated. (. . .) I have held some courses for teachers where many turn up because they feel like they have to. But they can see that they will not get enough time, and may be demotivated by it. And they do not really want to be there, even thought they want to learn, it is hard for them. Because they know that the one day not is enough to learn enough to bring it into the classroom.

Learning new concepts, like programming and debugging, can be demotivating and frustrating for teachers. Teacher δ also explain how the lack of knowledge on something to be taught in the classroom may lead to frustration for the teachers. The lack of time and learning material presents a challenge for the motivation for teachers to continue their professional development on computing aspects, including debugging.

A final result from the teacher interviews was that they have varying background and education in computing. Teacher ϵ brought this up when talking about constraints for professional development:

> [A constraint] is time, how much time we have to spend on professional development. And that we have varying degree of education in programming. Those two are the constraints. How much time and experience with programming one has. I think that is different for teachers.

The interviewed teachers also had very different background in learning computing, programming and debugging. Some had explicit education in teacher with ICT-teaching, some had experience from the professional field of ICT and some had learnt on their own due to interest in the field or necessity for teaching. A common factor is that none of the teachers had explicit teaching in debugging.

6.3 Implications

ID	**Learning objective**
A_3	The teacher should know of different types of bugs
B_3	The teacher should know how to use debugging strategies
C_3	The teacher should know how to promote and nurture a classroom culture for debugging
D_3	The teacher should know how to promote problem-solving and self-reliance in the pupils' debugging
E_3	The teacher should know how digital tools can aid debugging

Table 6.2: Learning objectives after the 3rd design cycle

In the 3rd design cycle no new learning objectives were added. The teachers evaluated the learning objectives from the 2nd design cycle, and the objectives were seen as fulfilling for debugging in the classroom. Learning objectives C_2 and D_2 were especially highlighted as beneficial for the development of the professional development of computing teachers. Based on the evaluation of the learning objectives with the teachers, no additions, removals or changes were made to the learning objectives in the 3rd design iteration.

Chapter 7

Discussion

The 3$^{\text{rd}}$ iteration of learning objectives, presented in table 6.2, represents the knowledge and skills teachers need to integrate debugging in their teaching of programming. The learning objectives are A_3) knowledge on different types of bugs, B_3) usage of debugging strategies, C_3) promotion of a culture for debugging, D_3) promotion of problem-solving and self-reliance and E_3) how digital tools can aid the debugging process. These learning objectives are grounded in the literature on debugging and debugging education and in the evaluation of experts and practitioners in the field of debugging education. Some of the learning objectives reflect previous studies from abroad, and they are placed in the context of the Norwegian computing classroom at secondary education level. A discussion with focus on each of the learning objectives is done in the following paragraphs.

A focus on what types of bugs there are, is the first of the learning objectives. This learning objective is designed to address different types of bugs requiring different approaches. While it is prevalent to not define the fixing of syntax and compile-time errors as debugging, Michaeli and Romeike (2019a) propose the inclusion of these types of errors in the K-12 programming classroom. This proposition is based on the impact syntax errors have on the computing happening in K-12 classrooms. This attention to the number of problems in the computing classroom that are caused by syntax and compile-time errors is also reflected in the teacher evaluation. The teachers expressed that compile-time and runtime errors that provide an error message are prominent in pupils programs. Solving these types of errors was also one of the difficulties defined by the teachers, and solving of these types of bugs takes up a noteworthy part of their teaching. Therefore, the teaching of how to recognise and handle these types of errors is an important focus in the computing classroom. For the teachers at lower secondary school, the transition from a block-based programming to a text-based programming language also calls for knowledge on different types of bugs. Where block-based programming language are designed to avoid syntax and semantic errors, text-based are not. It is central for teachers in this transition phase to guide pupils into the aspects and debugging of non-functional errors.

There are many different strategies for debugging present in the literature and the empirical data from the interviews. Some of the strategies presented at tertiary educational level, like test driven development and unit testing, are closer to the practice in the professional field of software development. Both in the expert and teacher evaluation it was expressed that the difference in programming in the professional field and the programming performed at secondary school are distinct in that school programming is less complex. Since the programs are shorter and less interdependent in a school setting, there is not a need for the more advanced debugging strategies. Debugging strategies that are suited for secondary education should be simple to match the complexity of school programs. Some examples of debugging strategies that may be suited for secondary computing education are presented in table 7.1.

Program chunking	Decomposition of program
Discrepancy detection	Descriptive variable and file names
Checking edge cases	Ordered structure of files
Forward tracing	Comments in code
Backward tracing	Pair programming
Printing of state	Pair debugging
Rubber ducking	Creating flowcharts
Breadth-first approach	

Table 7.1: Examples of strategies for debugging suited for secondary education

One of the learning objectives the teachers highlighted as promising was the nurturing of a classroom culture for debugging. This learning objective is partly designed to address unrealistic mindsets of writing programs without bugs on the first try. The teachers reported to the pupils having a lack of awareness on how big a part debugging is of programming. To create a classroom culture for debugging it will be important to implement debugging as an aspect of the computing classroom. By conducting activities like live coding or live debugging, the teacher may present a realistic representation of the programming experience. This also allows for teaching of debugging strategies or a systematic approach to debugging in a live and contextualised setting. The usage of debugging logs, as used by Chmiel and Loui (2004), could also be a way to set focus on debugging in the teaching and learning of programming. Explicitly teaching debugging is a common proposition from the literature review, and this would also set the focus on debugging as an integral part of programming and computing. The usage of collaborative debugging strategies like pair debugging or peer code review could also contribute to emphasising the debugging process. Lastly, to facilitate the need for mediation in the learning process, which is based in sociocultural learning theory, the PRIMM model may be used in design of exercises (Sentance et al., 2019).

The other learning objective the teachers focused on as an important aspect of debugging education was the promotion of problem-solving and self-reliance in the pupils' debugging. It is reported in both the literature and in the interviews with teachers that pupils often perform learned helplessness. Some of the pupils do not initiate the debugging process on their own accord. Teacher δ reported how this was something that separates the high achieving pupils and the pupils who struggle. This is also reported by McCauley et al. (2008) and Michaeli and Romeike (2019a). The findings of Chmiel and Loui (2004) show that debugging is a skill that can be taught, and it is not based on aptitude. The recommendation from the literature to overcome this learned helplessness is teaching of a systematic approach to debugging. Another approach to promote the self-reliance and problem-solving of pupils is a focus on acquisition of knowledge or solutions to similar problems, which is present in the literature and the interviews. The ability to search online while debugging was reported by two of the teachers as an important skill to learn for students to become self-reliant in the debugging process. According to the framework of Li et al. (2019), domain knowledge on how the programming language works, procedural knowledge on how to perform actions and previous experience are requirements for debugging. By searching for and reading documentation of the programming language and development environment, the domain and procedural knowledge may be expanded. Finding how others fixed similar problems to one's own builds on the previous experience of others. This may aid the understanding and fixing of a specific bug. The facilitation of online searches for documentation on programming language, examples of code snippets and solutions to similar problems may aid the pupil in developing the foundation of knowledge on which their debugging skills rest.

The teachers reported that pupils commonly request help when they encounter a bug in their programs without trying to debug the problem on their own first. An important part of their teaching was also to give hints of the debugging process by asking questions that guides the pupils in locating and fixing bugs. This is something the teachers reported to mostly do individually for the pupils, where they go from pupil to pupil. Michaeli and Romeike (2019b, p. 6) see this in connection with the concept of learned helplessness. By teaching a systematic approach to debugging, their study showed reduced amount of trial-and-error and fostering of self-reliance in the pupils' debugging. Since learned helplessness is reported to be present in the Norwegian computing classroom, it is relevant to battle the lack of self-reliance in the debugging of pupils in this context. As a systematic approach to debugging seemed to have effect in the German K-12 classroom studied by Michaeli and Romeike (2019b), this may be an appropriate and positive approach to dealing with learned helplessness in the context of this research project.

The final set of designed learning objectives reflects the required knowledges for debugging presented in the framework of Li et al. (2019). Learning objective A_3 is connected to domain knowledge. Domain knowledge in a debugging setting is knowing how the programming paradigm works and includes familiarity with the programming language. Knowledge on different types of bugs is an important part of understanding the function of a programming language, and the separation of different types of bugs is dependent on the programming language. This is most clearly displayed in the difference between a text-based programming language like JavaScript, and a block-based programming language like Scratch. Scratch is made to avoid syntax and compile-time errors (Maloney et al., 2004, pp. 106–107), whilst syntax is a central hurdle for pupils programming in text-based programming languages (McCauley et al., 2008, pp. 71–75). Therefore, the programming paradigm and the programming language represent the domain knowledge required to be able to debug. Knowledge on different types of bugs placed in the context of tool and programming language is therefore a part of the domain knowledge. The knowledges in the debugging framework are placed in the context of Norwegian secondary computing education. Learning objective B_3 is explicitly placed in the framework for debugging (Li et al., 2019). Strategic knowledge is about knowledge of debugging strategies. This is directly represented by learning objective B_3, which entails knowledge of debugging strategies. Learning objectives C_3 and D_3 are not placed in the content knowledge of debugging, but are rather pedagogical content knowledge or didactics of debugging. Since the framework is focused on the learner, and not the educator, these learning objectives are not present in the framework. The final learning objective E_3 is however represented in procedural knowledge. Procedural knowledge is about how to perform different debugging tasks and activities. In extension, this knowledge also entails knowledge of how to use the available tools to perform debugging tasks and activities. Therefore, knowing how the tools at hand can be used to aid in the debugging process is a part of the procedural knowledge of how to debug.

Like reported by other studies abroad (Fitzgerald et al., 2010, p. 390; Michaeli and Romeike, 2019a, p. 1030), the Norwegian teachers express that material is lacking on what the best practises for debugging education are. In the combination with the varying backgrounds and educations of teachers, learning material proves a condition for the continued development of teachers' debugging knowledge and skills. Resources should contain specification of what debugging in the computing classroom could and should entail. It should contain finished code snippets, classroom activities and teaching plans grounded in debugging didactics.

An unanimously reported constraint for the professional development of teachers' debugging knowledge is the lack of time. This constraint has implications for any implementation of the learning objectives. To accommodate the lack of time teachers have to spend on professional development, learning material that addresses the designed learning objectives should have a concise focus. Specific strategies and sequences with material that is close to the practice in the classroom should be created. Presenting practical activities the teacher can use directly in their teaching will be important to lessen the time consumption and give the teachers immediate resources to use in the computing classroom. Even though the learning objectives are aimed at teachers of both lower and upper secondary school, the implemented learning material should follow a specific subject curriculum. This to limit redundant information and to focus the material for the teaching context of the teacher.

Two of the conditions for the implementation of the learning objectives are the tools and programming languages used or available in a specific computing classroom. While there are common practices for what programming languages are used in lower and upper secondary computing classes, there are many different variations software and hardware used in classrooms. The tools and technologies can vary from school to school, subject to subject and pupil to pupil. Many of the teachers reported to work project centred and that they offered the pupils great freedom in choice of integrated development environment, libraries, project concept and implementation design. Due to the variations of used technologies, tools and libraries, any learning material based on the designed learning objectives should take technology into consideration. This can either be done by creating subject specific learning material as mentioned above, or by focusing on general functions that exists in most tools for development. Examples of such general aids for debugging process are colour coded keywords in the editor, the use of a debugger tool with breakpoints or an undo-button.

Just like there are variations of tools and technologies used in classrooms, the backgrounds and educations of teachers also vary. Some of the teachers in the study expressed confidence in their teaching of debugging and some expressed a desire to develop their debugging teaching. This may be a result of the different backgrounds with computing and the various educational routes. Some of the teachers have little to no education in programming, and the teachers reported to not have explicit education in debugging. Since debugging skills and knowledge seem to be a self-taught aspect of the computing of teachers, the teaching of debugging in classrooms necessarily also varies depending of the specific teacher. Different teachers may be at different stages in the development of the learning objectives presented as the result of the design cycles. There is also a difference between lower and upper secondary school. In upper secondary school text-based programming language is most prevalent, which include syntax errors. In lower secondary school block-based programming language is used mostly, which are designed to remove syntax errors. This creates a difference in the debugging conducted at the different levels of education, and therefore difference in what the teachers need to know to implement debugging as part of their programming teaching. This also proves a condition for learning objective E_3, since how a tool can aid the debugging process is as varied as the difference in programming tools and environments used in lower and upper secondary school.

The teachers reported that their teaching of debugging mostly happens prompted by pupils encountering bugs during programming. A systematic approach to debugging was not a part of the teaching, and few teaching sequences were explicitly about debugging. With regards to debugging being one of the factors that separates pupils who struggle and pupils who achieve highly in programming, this is an interesting find. If debugging is a threshold for achievement in programming subjects, there should be a focus on debugging in the computing education. The lack of explicit teaching of debugging is not the same as debugging missing from the education, but it could be beneficial to have a reflected and didactic approach to debugging in the teaching of programming. Li et al. (2019), Michaeli and Romeike (2019a) and Whalley et al. (2021) report to debugging being underrepresented in computing education in New Zealand and Germany. The results of this study suggests that this is not necessarily the case in Norway, but there is a lack of explicit focus on systematic debugging in the computing education. Such a focus may be relevant to address the common problem of learned helplessness. This is also in accordance with the recommendation of Li et al. (2019) and Michaeli and Romeike (2019a, 2019b, 2020) to teach a systematic approach to debugging. This may improve pupils' self-efficacy and debugging performance. A focus on systematic and explicit teaching of debugging may also be beneficial for the pupils who struggle with debugging, to advance them in their debugging and programming performance.

7.1 Limitations

The largest limitations of this research study are the sample size of informants and the size of the contents of the literature review. The research project is placed in the qualitative research paradigm, and to address the limitations in the sample sizes the context of the study is described in detail in chapter 1.3 and chapter 2, and a thick description of the method and results is provided. The analytical generalisation of the results and the thick description offers an insightful conclusion, but further research on a bigger sample size could be beneficial to validate the findings of the study and develop more general implementations of the learning objectives. The study also focuses on teachers of subjects with a sole focus on computing. Research with focus on teachers of subjects that only partly include computing (like mathematics, natural science, music and art and crafts) would also be insightful to the domain of debugging education and to generalise the results of this study to a wider context.

Chapter 8

Conclusion

The aim of this research project was to design learning objectives that can help computing teachers in Norwegian secondary school to conceptualise debugging education. The study has contributed to the field of debugging education in a few ways. The first contribution is an overview of concepts and didactics for teaching debugging as a part of programming subjects, in the form of learning objectives. The second contribution is implications for implementation of the learning objectives, based in conditions and constraints to teacher development. The findings are elaborated in the following, by answering the research questions.

8.1 What do teachers need to know?

> **RQ$_1$**: What do teachers need to know to integrate debugging in their teaching of programming?

RQ$_1$ is answered in the learning objectives designed during the research project. The learning objectives are presented in table 6.2, and contain two content objectives, two didactic objectives and one tool objective. The knowledge of different types of bugs is important to distinguish between different problems and the appropriate approach to solving them. This is especially important for teachers in lower secondary school, who have the responsibility to transition pupils from the block-based programming paradigm to the text-based programming paradigm. The teachers also need to know how to use different debugging strategies, like program chunking, backwards tracing, discrepancy detection and printing of state. The pedagogical objectives entail the classroom culture and self-reliance. To avoid pupil frustration and create a culture for debugging the teachers can use techniques like live coding, pair debugging or debugging logs. The most reported hurdle for pupils in computing classes in this study is learned helplessness. By teaching a systematic approach to debugging explicitly, the teachers may increase the self-reliance in the pupils' debugging and avoid pupils asking for help without trying to solve the bug on their own. The final learning objective entails knowledge on how digital tools may aid the debugging process. Since the availability and usage of tools and technologies vary greatly, the focus should lie on common aids like colour coding of keywords, breakpoints and undo functionalities.

8.2 What are the conditions and constraints?

> **$\mathbf{RQ_2}$**: What conditions and constraints are there for development of teachers' debugging skills?

The conditions and constraints give requirements and restrictions for the development of teachers' debugging skills. The main conditions for possible development are subject context and different professional or educational background of teachers. The difference in lower secondary school and upper secondary school is mostly based in programming language. While text-based programming is prominent in upper secondary school, lower secondary school mostly report to using block-based programming. This makes a big difference in the debugging processes. Also, within text-based programming pupils, teachers and schools use different tools and development environments. This also makes the needs for debugging strategies different for every teacher. The background a teacher has, both from professional computing settings and education in computing, also sets a condition for teacher development. Some teachers report to being comfortable in their debugging education, but most express a desire for additional development. The constraints for teacher development are mainly lack of time and lack of learning material. The field of computing is a rapidly developing one, and the need for teacher development is constant. The allotted time for professional development, including the development of debugging skills and knowledges, is restricted for teachers. In addition, there are no established best practices for debugging and a lack of learning material for debugging in the classroom. This study provides a conceptual overview of what debugging in computing education should entail, which is an important contribution towards learning material for debugging education in Norwegian secondary school.

Bibliography

Agile Alliance. (n.d.). *Pair Programming: Does it really work?* Retrieved 6th April 2022, from https://www.agilealliance.org/glossary/pairing

Chmiel, R. & Loui, M. C. (2004). Debugging: From novice to expert. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, USA*, 17–21. https://doi.org/10.1145/971300.971310

Directorate of Education and Training. (2020). *Læreplan i valgfaget programmering (PRG01-02)* [The elective Programming subject curriculum]. Established as regulation. The National Curriculum for Knowledge Promotion in Primary and Secondary Education. https://www.udir.no/lk20/prg01-02

Directorate of Education and Training. (2021). *Læreplan i informasjonsteknologi (INF01-02)* [Information technology subject curriculum]. Established as regulation. The National Curriculum for Knowledge Promotion in Primary and Secondary Education. https://www.udir.no/lk20/inf01-02

Dolonen, J. A., Kluge, A., Litherland, K. & Mørch, A. (2019). *Litteraturgjennomgang av programmering i skolen* [Literature review of programming in school]. University of Oslo. https://www.duo.uio.no/handle/10852/76290

Fitzgerald, S., McCauley, R., Hanks, B., Murphy, L., Simon, B. & Zander, C. (2010). Debugging from the student perspective. *IEEE Transactions on Education*, *53*(3), 390–396. https://doi.org/10.1109/TE.2009.2025266

Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00684ED1V01Y201511HCI033

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian Journal of Information Systems*, *19*(2), Article 4. http://aisel.aisnet.org/sjis/vol19/iss2/4

IKT i Stavangerskolen. (n.d.). *Programmering i skolen* [Programming in school]. Retrieved 28th May 2022, from https://sites.google.com/stavangerskolen.no/iktistavangerskolen/programmering

Lær kidsa koding. (n.d.). *Kurs* [Courses]. Retrieved 28th May 2022, from https://oppgaver.kidsakoder.no/

Li, C., Chan, E., Denny, P., Luxton-Reilly, A. & Tempero, E. (2019). Towards a framework for teaching debugging. *Proceedings of the Twenty-First Australasian Computing Education Conference, Australia*, 79–86. https://doi.org/10.1145/3286960.3286970

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. & Resnick, M. (2004). Scratch: A sneak preview. *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing, Japan*, 104–109. https://doi.org/10.1109/C5.2004.1314376

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L. & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67–92. https://doi.org/10.1080/08993400802114581

Michaeli, T. & Romeike, R. (2017). Addressing teaching practices regarding software quality: Testing and debugging in the classroom. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education, Netherlands*, 105–106. https://doi.org/10.1145/3137065.3137087

Michaeli, T. & Romeike, R. (2019a). Current status and perspectives of debugging in the K12 classroom: A qualitative study. *2019 IEEE Global Engineering Education Conference (EDUCON)*, 1030–1038. https://doi.org/10.1109/EDUCON.2019.8725282

Michaeli, T. & Romeike, R. (2019b). Improving debugging skills in the classroom: The effects of teaching a systematic debugging process. *Proceedings of the 14th Workshop in Primary and Secondary Computing Education, Scotland UK*, 7 pages. https://doi.org/10.1145/3361721.3361724

Michaeli, T. & Romeike, R. (2020). Investigating students' preexisting debugging traits: A real world escape room study. *Proceedings of the 20th Koli Calling International conference on computing education research, Finland*, 10 pages. https://doi.org/10.1145/3428029.3428044

Miljanovic, M. A. & Bradbury, J. S. (2017). RoboBUG: A serious game for learning debugging techniques. *Proceedings of the 2017 ACM Conference on International Computing Education Research, USA*, 93–100. https://doi.org/10.1145/3105726.3106173

Ministry of Education and Research. (2017). *Verdier og prinsipper for grunnopplæringen: Overordnet del av læreplanverket* [Values and principles for primary and secondary education: Core curriculum of the National Curriculum]. Established as regulation by royal resolution. The National Curriculum for Knowledge Promotion in Primary and Secondary Education. https://www.regjeringen.no/no/dokumenter/verdier-og-prinsipper-for-grunnopplaringen/id2570003/

Murphy, L., Fitzgeral, S., Hanks, B. & McCauley, R. (2010). Pair debugging: A transactive discourse analysis. *Proceedings of the Sixth International Workshop on Computing Education Research, Denmark*, 51–58. https://doi.org/10.1145/1839594.1839604

NOU 2015:8. (2015). *Fremtidens skole* [School of the future]. Ministry of Education and Research. https://www.regjeringen.no/no/dokumenter/nou-2015-8/id2417001/

NTNU. (n.d.). *Collection of personal data for research projects*. Retrieved 8th December 2021, from https://i.ntnu.no/wiki/-/wiki/English/Collection+of+personal+data+for+research+projects

Perscheid, M., Siegmund, B., Taeumel, M. & Hirschfeld, R. (2016). Studying the advancement in debugging practice of professional software developers. *Software Quality Journal*, *25*(1), 83–110. https://doi.org/10.1007/s11219-015-9294-2

Polit, D. F. & Beck, C. T. (2010). Generalization in quantitative and qualitative research: Myths and strategies. *Int J Nurs Stud*, *47*(11), 1451–1458. https://doi.org/10.1016/j.ijnurstu.2010.06.004

Postholm, M. B. (2005). *Kvalitativ metode: en innføring med fokus på fenomenologi, etnografi og kasusstudier* [Qualitative method: An introduction with focus on phenomenology, ethnography and case studies]. Universitetsforlaget.

Robson, C. & McCartan, K. (2016). *Real world research: A resource for users of social research methods in applied settings* (4th ed.). Wiley.

Säljö, R. (2000). *Lärande i praktiken: Ett sociokulturellt perspektiv.* [Learning in practice: A sociocultural perspective]. Prisma.

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). *Teknologi og programmering for alle: En faggjennomgang med forslag til endringer i grunnopplæringen - august 2016* [Technology and programming for everyone: A subject review with proposals for changes in the primary and secondary education and training - August 2016]. Directorate of Education and Training. https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/teknologi-og-programmering-for-alle/

Sentance, S., Waite, J. & Kallia, M. (2019). Teaching computer programming with PRIMM: A sociocultural perspective. *Computer science education*, *29*(2-3), 136–176. https://doi.org/10.1080/08993408.2019.1608781

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher*, *15*(2), 4–14. https://doi.org/10.3102/0013189x015002004

Siljan kodeklubb. (n.d.). *Programmering* [Programming]. Retrieved 28th May 2022, from https://siljankodeklubb.org/programmering

The Norwegian centre for ICT in education. (2017). *Programmering i skolen*. Directorate of Education and Training. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/notat-om-programmering-i-skolen/

Vygotsky, L. S. (2018). Interaksjon mellom læring og utvikling [Interaction between learning and development] (B. Christensen, Trans.). In E. L. Dale (Ed.), *Om utdanning: Klassiske tekster* (pp. 151–165). Gyldendal. (Original work published 1978).

Wang, X. & Souders, J. (2012). Improving debugging education through applied learning. *Journal of Computing Sciences in Colleges*, *27*(3), 138–145.

Whalley, J., Settle, A. & Luxton-Reilly, A. (2021). Novice reflections on debugging. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, USA*, 73–79. https://doi.org/10.1145/3408877.3432374

Appendix A

Interview guides for experts

Intervjuguide

Legg mobilen bort. Smil. Pust med magen.

Takk for at du stiller til intervju.

Masteroppgåva: Kva må lærarar kunne for å implementere debugging i undervisninga si?
Mål: Lage læringsmål og -aktivitetar som skal hjelpe lærarar til å utvikle det dei treng for å implementere debugging i undervisninga.

Dette er ein ekspertevaluering, byggjer på litteraturstudien. Ingen feile svar, berre ei evaluering og utfylling av mitt arbeid.

Kvifor har eg velt deg? Fagkunnskap, faginnhald og undervisningserfaring/informatikkdidaktikk og kjennskap til utdanning av komande programeringslærarar/opplæring av programmeringslærarar og kjennskap til tinga si tilstand i dag.

Det står om opptaket i samtykkeskjema, men det viktigaste er at lydfila slettast etter prosjektet er ferdig og at du kan velje å bli anonymisert.

feilsøking = debugging
programfeil/feil i koda = bugs

Har du nokre spørsmål?

Læringsmål

A. Læraren skal kjenne til ulike typar programfeil.
B. Læraren skal kunne bruke globale og lokale feilsøkingsstrategiar.
    a. Oppdeling av program
    b. Avvik i inndata/utdata
    c. Lesing av program
    d. Samarbeidande utvikling
    e. Førebygging mot programfeil
C. Læraren skal vite kvifor feilsøking er ein viktig del av informatikkundervisning.
D. Læraren skal kunne nokre teknikkar for å fremme problemløysing og sjølvstendigheit hos elevane.
E. Læraren skal kunne stille tilbake eit mislykka forsøk på å rette opp ein programfeil.

Learning objectives

A. The teacher should have knowledge about different types of bugs.
B. The teacher should know how to use some global and local debugging strategies.
    a. Program chunking
    b. Discrepancy detection
    c. Program tracing
    d. Collaborative development
    e. Bug prevention
C. The teacher should have an understanding of why debugging is an important part of computing education.
D. The teacher should know some techniques for promoting problem-solving and self-reliance in the pupils.
E. The teacher should know how to revert unsuccessful attempts at fixing bugs.

Introkurs for studentar

1. Er debugging ein del av undervisninga di?
    a. Har du døme på korleis debugging er implementert i undervisninga di?
    b. Omtrent kor mange timar brukar du på det?
2. Har du døme på eit konsept knytt til debugging som studentar opplev som vanskeleg?
3. Har du døme på ein læringsaktivitet knytt til debugging har vore bra for studentane sin læring?
4. Ser du skilnad/endring på studentar sine ferdigheiter innan debugging?
    a. Mellom ulike studentar?
    b. Etter kvart som dei lærer meir om programmering?
5. Kan du nemne nokre strategiar du nyttar for debugging i undervisninga di?
    a. Samarbeid?
6. *Vis læringsmåla.*
7. Dersom ein fyller opp læringsmåla, har ein då alt ein treng for å kunne integrere debugging i programmeringsundervisninga si?

Kurs for lærarar

1. Er debugging ein del av undervisninga di?
    a. Har du eit døme på korleis det er implementert? Innhald?
    b. Omtrent kor mange timar bukar du på det?
2. Har du døme på eit konsept knytt til debugging som lærarar opplev som vanskeleg?
3. Har du døme på ein læringsaktivitet knytt til debugging har vore bra for studentane sin læring?
4. Ser du skilnad/endring på lærarar sine ferdigheiter innan debugging?
    a. Mellom ulike studentar basert på tidlegare erfaring?
    b. Etter kvart som dei lærer meir om programmering?
5. Har du døme på ein strategi for debugging mange lærarar nyttar?
6. Kva er hovudutfordringa for lærarar som tek kurset ditt, med tanke på undervisning av programmering?
    a. Kan debugging vere eit verktøy for å dekke noko av problematikken?
7. *Vis læringsmåla.*
8. Dersom ein fyller opp læringsmåla, har ein då alt ein treng for å kunne integrere debugging i programmeringsundervisninga si?
9. Dersom ein skal fylle læringsmåla, har du nokre tankar om tid og/eller format?

Informatikkdidaktikk

1. Er debugging ein del av undervisninga di?
    a. Har du eit døme på korleis det er integrert?
    b. Omtrent kor mange timar brukar du på det?
2. Har du døme på eit konsept knytt til debugging som lærarstudentar opplev som vanskeleg?
3. Har du døme på ein læringsaktivitet knytt til debugging har vore bra for lærarstudentane sin læring?
4. Har du døme på korleis samarbeid kan bli nytta for å fremje lærarstudentar si læring (av debugging)?
5. Korleis er stoda i dag med både programmeringsferdigheiter og informatikkdidaktikk, både hos lærarar og hos lærarstudentar?
    a. Kva er hovudutfordringa?
    b. Kva er ein læringsmetode/-aktivitet som har støtta utviklinga?
6. *Vis læringsmåla.*
7. Dersom ein fyller opp læringsmåla, har ein då alt ein treng for å kunne integrere debugging i programmeringsundervisninga si?
8. Dersom ein skal fylle læringsmåla, har du nokre tankar om tid og/eller format?

Appendix B

Interview guide for teachers

Intervjuguide

Legg mobilen bort. Smil. Pust med magen.

Takk for at du stiller til intervju.

Masteroppgåva: Kva må lærarar kunne for å implementere debugging i undervisninga si?
Mål: Lage læringsmål og -aktivitetar som skal hjelpe lærarar til å utvikle det dei treng for å implementere debugging i undervisninga.

Dette er ein ekspertevaluering, byggjer på litteraturstudien. Ingen feile svar, berre ei relevas og utfylling av mitt arbeid.

Kvifor har eg velt deg? Fagkunnskap, faginnhald og undervisningserfaring/informatikkdidaktikk og kjennskap til utdanning av komande programeringslærarar/opplæring av programmeringslærarar og kjennskap til tinga si tilstand i dag.

Det står om opptaket i samtykkeskjema, men det viktigaste er at lydfila slettast etter prosjektet er ferdig og at du kan velje å bli anonymisert.

feilsøking = debugging

programfeil/feil i koda = bugs

Har du nokre spørsmål?

- *Forklar kva debugging er.*
- *Vis fram læringsmål + nokre aktivitetar.*
- Er dette eit fullverdig sett med læringsmål for dundervisning av debugging?
    - Finst det andre utfordringar i klasserommet med omsyn på debugging?
    - Finst det andre strategiar eller prosessar for debugging i klasserommet?
- Kan du gje eit døme korleis aktivitetar som kan bidra til å nå læringsmåla?
- Finst det nokre grensande faktorar eller krav som må vere på plass for at lærarar skal kunne nå desse læringsmåla?

Læringsmål

A. Læraren skal kjenne til ulike typar programfeil.
   - Syntaks, semantisk, logisk
   - Compile-time, runtime, funksjonell

B. Læraren skal kunne bruke feilsøkingsstrategiar.
   - Kommentere ut kode
   - Manuell sjekking av kjente verdiar
   - Manuell sjekking av ytterpunkt
   - Lese koda framover
   - Lese koda bakover
   - Sjå tilstand og variabelinnhald
   - Rubberducking
   - Breidde først
   - *Dele program i funksjonar og filer*
   - *Fornuftige namn på variablar og filer*
   - *Fornuftig filstruktur*
   - *Kommentering av kode*
   - *Parprogrammering*
   - *Teikning av modell*

C. Læraren skal kunne fremme og halde ved like ein klasseromskultur for feilsøking.
   - Live debugging
   - Pair debugging
   - Refleksjon over debugging
   - Debugging-logg

D. Læraren skal kunne fremme problemløysing og sjølvstendigheit i elevane si feilsøking.
   - Live koding
   - Live debugging
   - PRIMM
   - Parprogrammering
   - Refleksjonslogg
   - Debug-logg
   - Haldningsbygging

E. Læraren skal kjenne til korleis digitale verktøy kan gje stø i feilsøkingsprosessen.
   - Debugger-verktøy
     - Køyre program linje-for-linje
     - Breakpoint
     - Sjå variabelinnhald
   - Markeringar i IDE
     - Syntaktiske feil
     - Semantiske feil
     - Fargar på nøkkelord

Læringsmål

A. Læraren skal kjenne til ulike typar programfeil.
B. Læraren skal kunne bruke feilsøkingsstrategiar.
C. Læraren skal kunne fremme og halde ved like ein klasseromskultur for feilsøking.
D. Læraren skal kunne fremme problemløysing og sjølvstendigheit i elevane si feilsøking.
E. Læraren skal kjenne til korleis digitale verktøy kan gje stø i feilsøkingsprosessen.

Appendix C

Consent form for experts

[redacted]

Trondheim, 7. mars 2022

Vil du delta i forskingsprosjektet

"Debugging i norsk skule"?

Dette er eit spørsmål til deg om å delta i eit forskingsprosjekt der formålet er å samle kva debugging i undervisning skal innebere og korleis dette passar inn i konteksten av norsk ungdomsskule og vidaregåande skule. I dette skrivet gjev eg deg informasjon om måla for prosjektet og om kva deltaking vil innebere for deg.

Formål
Formålet med fordjupingsprosjektet er å samle konsept og prosedyrar innan debugging som burde vere ein del av programmeringsundervisninga og -didaktikken i Noreg. Eit første utval av læringsmål vil vere basert på forsking og litteratur om temaet, og deretter evaluert og utfylt av ei ekspertgruppe. Målet er å formulere læringsmål for programmeringslærarar i norsk ungdomsskule og vidaregåande skule, om kva dei burde fokusere på for å implementere debugging på ein didaktisk måte i sine klasserom. Dette prosjektet vil dermed bestå av intervju med undervisarar i programmering ved norske universitet og høgskuler, som vil funke som ekspertar på debugging i ein norsk kontekst.

Kven er ansvarleg for forskingsprosjektet?
Institutt for datateknologi og informatikk ved Noregs teknisk-naturvitskaplege universitet (NTNU) er ansvarleg for prosjektet.

Kvifor får du spørsmål om å delta?
Du får spørsmål om å delta sidan du har erfaring med å undervise programmeringsrelaterte emne ved universitet eller høgskule. Formålet med forskingsstudiet er å produsere fullverdige og relevante læringsmål for programmeringslærarar, og du vil dermed gje viktig innsyn i kva undervisning av debugging i ein norsk kontekst handlar om, samt utfordringar og moglegheiter ved undervisning av debugging. Eiga nettverk er nytta for å ta kontakt med deg.

Kva inneber det for deg å delta?
Viss du vel å delta i prosjektet, inneber det at du deltek på eit intervju på 20-30 minutt. Intervjuet vil bli teke opp med ekstern bandopptakar, og transkriberast for å analysere og diskutere forskingsspørsmåla. Temaa for intervjuet vil vere drøfting av presenterte læringsmål om debugging for lærarar, samt eventuelle refleksjonar eller erfaringar som kan bidra til å utvikle didaktikken rundt temaet.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Dersom du vel å delta, kan du når som helst trekkje samtykket tilbake utan å gje nokon grunn. Alle personopplysingane dine vil då bli sletta. Det vil ikkje føre til nokon negative konsekvensar for deg dersom du ikkje vil delta eller seinare vel å trekkje deg.

Ditt personvern – korleis vi oppbevarer og bruker opplysingane dine
Vi vil berre bruke opplysingane om deg til formåla vi har fortalt om i dette skrivet. Vi behandlar opplysingane konfidensielt og i samsvar med personvernregelverket. Forskar/student vil ha tilgang til lydopptaket frå intervjuet. Dersom du samtykker til å ikkje stille anonymt, kan namnet og stillingstittelen din bli presentert i rapporten av forskingsprosjektet. Viss ikkje, vil berre anonymiserte sitat frå transkripsjonane bli nytta. Opptaket vil bli transkribert og anonymisert, der kvar deltakar får eit alias, slik at identifikasjon av deltakaren ikkje vil vere mogleg. Dataa vil bli lagra på ein passordbeskytta server ved NTNU.

Kva skjer med opplysingane dine når vi avsluttar forskingsprosjektet?
All dataa vil bli destruert etter at masterprosjektet er gjennomførte, som etter planen vil vere 1. august 2022.

Kva gjev oss rett til å behandle personopplysingar om deg?
Vi behandlar opplysingar om deg basert på samtykket ditt.

På oppdrag frå NTNU har NSD – Norsk senter for forskningsdata AS vurdert at behandlinga av personopplysingar i dette prosjektet er i samsvar med personvernregelverket.

Dine rettar
Så lenge du kan identifiserast i datamaterialet, har du rett til:
- innsyn i kva opplysingar vi behandlar om deg, og å få utlevert ein kopi av opplysingane,
- å få retta opplysingar om deg som er feil eller misvisande,
- å få sletta personopplysingar om deg,
- å sende klage til Datatilsynet om behandlinga av personopplysingane dine.

Dersom du har spørsmål til studien, eller om du ønskjer å vite meir eller utøve rettane dine, ta kontakt med:
- Institutt for datateknologi og informatikk ved prosjektansvarleg [redacted]
- Vårt personvernombod: [redacted]

Dersom du har spørsmål knytt til NSD si vurdering av prosjektet kan du ta kontakt med:
- NSD – Norsk senter for forskningsdata AS, på e-post (personverntjenester@nsd.no) eller på telefon: 55 58 21 17.

Venleg helsing

[redacted]
(Forskar/rettleiar)

[redacted]
Student

Samtykkeerklæring

Eg har motteke og forstått informasjon om prosjektet ”Debugging i norsk skule” og har fått høve til å stille spørsmål.

☐ Eg samtykker til å delta i intervju.

☐ Eg samtykker til at opplysingane mine kan behandlast fram til prosjektet er avslutta.

☐	Eg samtykker til at anonymiserte sitat kan presenterast i rapporten.	eller	☐	Eg samtykker til at mitt namn og tilsettingstittel kan presenterast i rapporten.

(Signert av prosjektdeltakar, dato)

Appendix D

Consent form for teachers

[redacted]

Trondheim, 27. april 2022

Vil du delta i forskingsprosjektet *"Debugging i norsk skule"*?

Dette er eit spørsmål til deg om å delta i eit forskingsprosjekt der formålet er å samle kva debugging i undervisning skal innebere og korleis dette passar inn i konteksten av norsk ungdomsskule og vidaregåande skule. I dette skrivet gjev eg deg informasjon om måla for prosjektet og om kva deltaking vil innebere for deg.

Formål
Formålet med fordjupingsprosjektet er å samle konsept og prosedyrar innan debugging som burde vere ein del av programmeringsundervisninga og -didaktikken i Noreg. Eit første utval av læringsmål vil vere basert på forsking og litteratur om temaet, og deretter evaluert og utfylt av ei ekspertgruppe og undervisarar. Målet er å formulere læringsmål for programmeringslærarar i norsk ungdomsskule og vidaregåande skule, om kva dei burde fokusere på for å implementere debugging på ein didaktisk måte i sine klasserom. Dette prosjektet vil dermed bestå av intervju med undervisarar i programmering ved ungdomsskulen og vidaregåande skule.

Kven er ansvarleg for forskingsprosjektet?
Institutt for datateknologi og informatikk ved Noregs teknisk-naturvitskaplege universitet (NTNU) er ansvarleg for prosjektet.

Kvifor får du spørsmål om å delta?
Du får spørsmål om å delta sidan du har erfaring med å undervise programmering ved ungdomsskule eller vidaregåande. Formålet med forskingsstudiet er å produsere fullverdige og relevante læringsmål for programmeringslærarar, og du vil dermed gje viktig innsyn i kva undervisning av debugging i ein norsk skule handlar om, samt utfordringar og moglegheiter til undervisning av debugging. Eiga nettverk er nytta for å ta kontakt med deg.

Kva inneber det for deg å delta?
Viss du vel å delta i prosjektet, inneber det at du deltek på eit intervju på omtrent 30 minutt. Intervjuet vil bli teke opp med ekstern bandopptakar, og transkriberast for å analysere og diskutere forskingsspørsmåla. Temaa for intervjuet vil vere drøfting av presenterte læringsmål om debugging for lærarar, samt eventuelle refleksjonar eller erfaringar som kan bidra til å utvikle didaktikken rundt temaet.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Dersom du vel å delta, kan du når som helst trekkje samtykket tilbake utan å gje nokon grunn. Alle personopplysingane dine vil då bli sletta. Det vil ikkje føre til nokon negative konsekvensar for deg dersom du ikkje vil delta eller seinare vel å trekkje deg.

Ditt personvern – korleis vi oppbevarer og bruker opplysingane dine
Vi vil berre bruke opplysingane om deg til formåla vi har fortalt om i dette skrivet. Vi behandlar opplysingane konfidensielt og i samsvar med personvernregelverket. Forskar/student vil ha tilgang til

lydopptaket frå intervjuet. Berre anonymiserte sitat frå transkripsjonane bli nytta. Opptaket vil bli transkribert og anonymisert, der kvar deltakar får eit alias, slik at identifikasjon av deltakaren ikkje vil vere mogleg. Dataa vil bli lagra på ein passordbeskytta server ved NTNU.

Kva skjer med opplysingane dine når vi avsluttar forskingsprosjektet?
All dataa vil bli destruert etter at masterprosjektet er gjennomførte, som etter planen vil vere 1. august 2022.

Kva gjev oss rett til å behandle personopplysingar om deg?
Vi behandlar opplysingar om deg basert på samtykket ditt.

På oppdrag frå NTNU har NSD – Norsk senter for forskningsdata AS vurdert at behandlinga av personopplysingar i dette prosjektet er i samsvar med personvernregelverket.

Dine rettar
Så lenge du kan identifiserast i datamaterialet, har du rett til:

- innsyn i kva opplysingar vi behandlar om deg, og å få utlevert ein kopi av opplysingane,
- å få retta opplysingar om deg som er feil eller misvisande,
- å få sletta personopplysingar om deg,
- å sende klage til Datatilsynet om behandlinga av personopplysingane dine.

Dersom du har spørsmål til studien, eller om du ønskjer å vite meir eller utøve rettane dine, ta kontakt med:

- Institutt for datateknologi og informatikk ved prosjektansvarleg [redacted]
- Vårt personvernombod: [redacted]

Dersom du har spørsmål knytt til NSD si vurdering av prosjektet kan du ta kontakt med:

- NSD – Norsk senter for forskningsdata AS, på e-post (personverntjenester@nsd.no) eller på telefon: 55 58 21 17.

Venleg helsing

[redacted]
(Forskar/rettleiar)

[redacted]
Student

Samtykkeerklæring

Eg har motteke og forstått informasjon om prosjektet "Debugging i norsk skule" og har fått høve til å stille spørsmål.

- Eg samtykker til å delta i intervju.
- Eg samtykker til at opplysingane mine kan behandlast fram til prosjektet er avslutta.

(Signert av prosjektdeltakar, dato)

Appendix E

Assessment of NSD part 1

Meldeskjema / Debugging i norsk skule / Vurdering

Vurdering

Referansenummer
216673

Prosjekttittel
Debugging i norsk skule

Behandlingsansvarlig institusjon
Norges teknisk-naturvitenskapelige universitet / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for datateknologi og informatikk

Prosjektperiode
10.01.2022 - 31.08.2022

Meldeskjema

Dato	**Type**
17.03.2022	Standard

Kommentar
OM VURDERINGEN
Personverntjenester har en avtale med institusjonen du forsker eller studerer ved. Denne avtalen innebærer at vi skal gi deg råd slik at behandlingen av personopplysninger i prosjektet ditt er lovlig etter personvernregelverket.

Personverntjenester har nå vurdert den planlagte behandlingen av personopplysninger. Vår vurdering er at behandlingen er lovlig, hvis den gjennomføres slik den er beskrevet i meldeskjemaet med dialog og vedlegg.

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til den datoen som er oppgitt i meldeskjemaet.

LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra de registrerte til behandlingen av personopplysninger. Vår vurdering er at prosjektet legger opp til et samtykke i samsvar med kravene i art. 4 og 7, ved at det er en frivillig, spesifikk, informert og utvetydig bekreftelse som kan dokumenteres, og som den registrerte kan trekke tilbake.

Lovlig grunnlag for behandlingen vil dermed være den registrertes samtykke, jf. personvernforordningen art. 6 nr. 1 bokstav a.

PERSONVERNPRINSIPPER
Personverntjenester vurderer at den planlagte behandlingen av personopplysninger vil følge prinsippene i personvernforordningen om:

- lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at de registrerte får tilfredsstillende informasjon om og samtykker til behandlingen
- formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke behandles til nye, uforenlige formål
- dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet
- lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

DE REGISTRERTES RETTIGHETER
Så lenge de registrerte kan identifiseres i datamaterialet vil de ha følgende rettigheter: innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18), og dataportabilitet (art. 20).

Personverntjenester vurderer at informasjonen om behandlingen som de registrerte vil motta oppfyller lovens krav til form og innhold, jf. art. 12.1 og art. 13.

Vi minner om at hvis en registrert tar kontakt om sine rettigheter, har behandlingsansvarlig institusjon plikt til å svare innen en måned.

FØLG DIN INSTITUSJONS RETNINGSLINJER
Personverntjenester legger til grunn at behandlingen oppfyller kravene i personvernforordningen om riktighet (art. 5.1 d), integritet og konfidensialitet (art. 5.1. f) og sikkerhet (art. 32).

Ved bruk av databehandler (spørreskjemaleverandør, skylagring eller videosamtale) må behandlingen oppfylle kravene til bruk av

Ved bruk av databehandler (spørreskjemaleverandør, skylagring eller videosamtale) må behandlingen oppfylle kravene til bruk av databehandler, jf. art 28 og 29. Bruk leverandører som din institusjon har avtale med.

For å forsikre dere om at kravene oppfylles, må dere følge interne retningslinjer og/eller rådføre dere med behandlingsansvarlig institusjon.

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til oss ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde: https://www.nsd.no/personverntjenester/fylle-ut-meldeskjema-for-personopplysninger/melde-endringer-i-meldeskjema

Du må vente på svar fra oss før endringen gjennomføres.

OPPFØLGING AV PROSJEKTET
Personverntjenester vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

Lykke til med prosjektet!

Appendix F

Assessment of NSD part 2

Meldeskjema / Debugging i norsk skule / Vurdering

Vurdering

Referansenummer
216673

Prosjekttittel
Debugging i norsk skule

Behandlingsansvarlig institusjon
Norges teknisk-naturvitenskapelige universitet / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for datateknologi og informatikk

Prosjektperiode
10.01.2022 - 31.08.2022

Meldeskjema

Dato	**Type**
18.05.2022	Standard

Kommentar
Personverntjenester har vurdert endringen registrert i meldeskjemaet.

Det er vår vurdering at behandlingen av personopplysninger i prosjektet vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet med vedlegg. Behandlingen kan fortsette.

ENDRING
Det er lagt til et nytt utvalg 2.

LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra de registrerte til behandlingen av personopplysninger. Vår vurdering er at prosjektet legger opp til et samtykke i samsvar med kravene i art. 4 og 7, ved at det er en frivillig, spesifikk, informert og utvetydig bekreftelse som kan dokumenteres, og som den registrerte kan trekke tilbake.

Lovlig grunnlag for behandlingen vil dermed være den registrertes samtykke, jf. personvernforordningen art. 6 nr. 1 bokstav a.

DE REGISTRERTES RETTIGHETER
Så lenge de registrerte kan identifiseres i datamaterialet vil de ha følgende rettigheter: innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18), og dataportabilitet (art. 20).

Personverntjenester vurderer at informasjonen om behandlingen som de registrerte vil motta oppfyller lovens krav til form og innhold, jf. art. 12.1 og art. 13.

Vi minner om at hvis en registrert tar kontakt om sine rettigheter, har behandlingsansvarlig institusjon plikt til å svare innen en måned.

OPPFØLGING AV PROSJEKTET
Vi vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

Kontaktperson: [redacted]
Lykke til videre med prosjektet!