Master's thesis

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

Niklas Joakim Nystad

Formative Assessment and Code Reuse

Teachers' experiences, challenges and best practices in programming classes

Master's thesis in Natural Science with Teacher Education
Supervisor: Monica Divitni and co-supervisor Majid Rouhani
June 2020

NTNU
Norwegian University of Science and Technology

Niklas Joakim Nystad

Formative Assessment and Code Reuse

Teachers' experiences, challenges and best practices in programming classes

Master's thesis in Natural Science with Teacher Education
Supervisor: Monica Divitni and co-supervisor Majid Rouhani
June 2020

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

NTNU

Kunnskap for en bedre verden

Abstract

Knowledge, software, and code reuse are common and accepted ways of programming. For teachers this proves a challenge when it comes to assessment. What is the learners knowledge and skill, and what is straight up copy and paste? The use of programming has exploded the past decades, and more and more countries integrate programming and coding in formal education in distinct IT subjects as well as in natural sciences and other academic disciplines. This project has aimed at discovering how code reuse in programming education can relate to formative assessment as a practice when teaching and learning, and proposed some best practices to work formative in an appropriate and efficient manner. When programming, reusing problem solving, software, programs and particularly code is a common, efficient and recommended way of writing new programs and developing new software. A specific didactic topic of interest - formative assessment - is proven to have great impact on learning, and working formative has changed the way teachers work and consider learning. In this research project I have aimed at answering the problem statement "What are teachers' experiences with formative assessment with regard to code reuse in programming education?". I have conducted a grounded theory interview study, asking teachers about their attitudes towards code reuse in programming education, their challenges with formative assessment and best practices with regards to code reuse. This topic in teaching lacks guidelines, leaving it to the individual teacher whether they support or reject the concept of reuse, and attitudes and approaches towards code reuse in the classroom span wide. Some teachers are very open and positive to include it as a practice, while others are more restrictive and prefer that learners avoid it. I have found challenges revolving around code reuse as a practice on expense of proper understanding and training of general and specific programming skills. Finally, I have found and presented some of the teachers' best practices to overcome these challenges. This includes various formative assessment practices, appropriate exercise design, general formative teaching practice, awareness and teaching of code reuse and more a way of assessing code and programs. The research project shows that code reuse is indeed an interesting field of research also in education, and that studying code reuse in relation to specific didactic topics such as formative assessment, can gain valuable knowledge about how we are teaching programming, and what we can do to increase learning gains.

Sammendrag

Å programmere med kunnskaps-, programvare- og kodegjenbruk er vanlig og akseptert. For lærere blir dette en utfordring når det kommer til vurdering. Hva kan betraktes som elevers kunnskap og ferdigheter, og hva er ren klipping og liming? Bruken av programmering har eksplodert de siste tiårene, og fler og fler land integrerer programmering og koding i formell utdanning i distinkte IT-fag, i tillegg til i realfag og andre akademiske retninger. Dette prosjekter sikter seg inn på å finne ut hvordan kodegjenbruk i programmeringsundervisning kan relateres til formativ vurdering som undervisnings- og læringspraksis. I tillegg sikter den seg på å finne anbefalt praksis for å arbeide formativt på en hensiktsmessig og effektiv måte. Når en programmerer er gjenbruk av problemløsning, programvare, programmer og spesifikt kode både en vanlig, effektiv og anbefalt måte å skrive nye programmer, og utvikle ny programvare. Et konkret interessant didaktisk tema - formativ vurdering - har beviselig stor innvirkning på læring, og å arbeide formativt har endret måten lærere underviser og betrakter læring. I dette forskningsprosjektet har jeg siktet på å svare problemstillingen "Hva er læreres erfaringer med formativ vurdering med hensyn på kodegjenbruk i programmeringsundervisning?". Jeg har gjennomført en grounded theory intervjustudie, og spurt lærere om deres holdninger til kodegjenbruk i programmeringsundervisning, deres utfordringer med formativ vurdering samt deres anbefalinger til praksis med hensyn på kodegjenbruk i undervisningen. Dette undervisningstemaet mangler retningslinjer, og lar det være opp til den individuelle lærer hvorvidt de støtter eller avviser gjenbruk som konsept. I tillegg er holdningene til kodegjenbruk i klasserommet spredt. Noen lærere er svært åpne og positive til å inkludere det som praksis, imens andre er mer restriktive og foretrekker at elevene unngår det. Jeg har funnet utfordringer rundt kodegjenbruk som praksis på bekostning av ordentlig forståelse og trening i generelle og spesifikke programmeringsferdigheter. Til slutt har jeg funnet og presentert noen av lærernes anbefalinger til praksis for å mestre disse utfordringene. Disse inkluderer ulike formative praksiser, hensiktsmessige oppgavedesign, generell formativ undervisningspraksis, bevissthet rundt å vurdere og lære kodegjenbruk, og mer en måte å vurdere kode og programmer. Forskningsprosjektet viser at kodegjenbruk faktisk er et interessant forskningsområde også innen utdanning, og at å forske på kodegjenbruk i sammenheng med spesifikke didaktiske temaer slik som formativ vurdering kan ha stor nytteverdi med hensyn på hvordan vi underviser programmering, og hva vi kan gjøre for å øke læringsutbyttet.

Preface

This thesis ends several years of studying in order to work as a teacher. I want to thank my supervisor Monica Divitini who always knew when to tell me my work was sloppy, and when I needed to hear that everything is good and I should just keep going. You have provided professional, appropriate and dearly helpful guidance and feedback throughout the past year. A thank you as well to my co-supervisor Majid Rouhani for second opinions and discussions considering contributions of programming didactics research. A special thanks to all the teachers participating in providing the data through interviews, making this research project possible. You have given me amazing insights into how to teach programming, how to assess learners and how you experience your daily work as teachers. You do an invaluable job teaching young people difficult subjects, keeping them motivated and focused. I would also like to thank my fellow students and colleagues who during a very difficult time of isolation and uncertainty have joined social and professional meetings online, helped with the project and kept me working towards finishing this thesis. A final thank you to my partner who let me use the living room as an office for such a long period of time.

Contents

List of Figures

1 Introduction

In this research project, I have researched teachers' experiences with formative assessment in programming education in relation to code reuse as a programming practice. I have explored how teachers include code reuse when they teach programming, if there are any special challenges with formative assessment because of code reuse, and what kind of practice they recommend in order to overcome these challenges and work with formative assessment in programming education.

Information and communication technology, computer science, and programming in particular are academic disciplines which have increased massively in popularity in recent years, and are still growing. IT is integrated in a vast amount of businesses, and an increasing amount of people are studying it at colleges and universities. In the last few years primary and secondary schools have also started integrating programming both as a part of various subjects and courses, and also as a subject itself. In Norwegian secondary schools, programming has been a subject in experimental courses (Utdanningsdirektoratet, 2016, 2017), and with the new curriculum *Fagfornyelsen* it will be implemented in IT subjects as well as several others, such as natural sciences and mathematics (Utdanningsdirektoratet, n.d.-a). Norwegian primary and secondary schools are collaborating with programming and coding initiatives for children such as *Lær Kidsa Koding* to teach younger learners programming (*Lær Kidsa Koding*, n.d.). Because programming is a relatively new subject in schools, we desperately need more research in IT didactics to fully understand how we should teach coming generations IT and programming.

Assessment is probably something teachers do every day both intentionally and without intention, through evaluating learners' knowledge, behaviour and social skills. Assessment and evaluation happens all the time, by parents of children, at a work place or between friends, but public debate around assessment has mostly been focused around school and education. Assessment is a widely debated topic both pedagogically and politically, in Norway as well as several other countries (Engh, 2017, p.17). Teachers report that working with assessment is difficult and daunting, and a lot of resources are used to create guidelines and tools helping teachers to assess learners in an appropriate way (See for instance ILU, n.d., Resources Assessment for Learning). Formative assessment has also been proven in several studies to have a great impact on learning (Black, 1998; Black & William, 1998; Black, Harrison, Lee, Marshall, & Wiliam, 2004; Black & Wiliam, 2009b; Engh, 2017). Formative assessment is assessment practice with the intention of creating positive changes for the further learning (Black & William, 1998; Engh, 2017). Assessment is formative when the information, knowledge and practices are used to adopt the teaching work to meet the learning needs and promote learners' learning (Black & William, 1998; Black, 1998; Engh, 2017). The need for research in IT didactics also raises a need for teachers to learn about how they can work with formative assessment particularly in programming education.

1.1 Problem

During the fall of 2019 I conducted a research interview study in a research method course at NTNU, where I interviewed teachers teaching programming in high school about their experiences and challenges with assessment when teaching programming. A topic that frequently came up was the use of external resources when teaching. Using external resources, the teachers described, could mean the learners using someone else's code snippets, repositories and programs as their own. It could also be searching online for known patterns and solutions to programming tasks and problems. In addition, it could mean finding, downloading and using big, complex and extensive resources and put them together with their own work to make something function properly. The teachers discussed how assessing the learners could be complicated when they downloaded other people's code, programs and solutions, because it became more difficult to know if the learners had actually learned, if the learners truly understood the problems and if the reuse of code and programs was legitimate or should be considered plagiarism. On the other hand, the learners were able to work with more interesting and complex problems in the classroom when all parts of the solutions didn't need to be made by themselves. In addition, they pointed out how reuse as a concept is an essential part of programming.

A common way of working with programming on a professional level includes the use of external resources and reuse of existing solutions in various ways. The process is called software reuse (Sojer, 2011). Ask any working software developer and they will likely confirm that a huge part of everyday programming tasks consists of searching for solutions, guides, tools, error messages and help on the internet. When programming, it is perfectly normal to reuse other people's code such as prepared algorithms and functional objects, or to import full libraries or complete open-source repositories to do some specific task (Sojer, 2011; Frakes & Kang, 2005). In the coming Norwegian curriculum for Information Technology courses in high school, new learning goals will also more explicitly include code reuse than before (Utdanningsdirektoratet, 2020a)

Software, programs and code, or knowledge about them, can be used over and over in new software, programs and code for similar or new tasks. Reusing code is a common way of programming to save time and resources, to work more efficiently, to use known useful and robust "building blocks" in programs, and to simply not "re-invent the wheel" every time a programming task is to be done (Frakes & Kang, 2005; Umarji & Sim, 2013). Code reuse is the process of using existing code or code knowledge to create or develop new code, software or programs, rather than creating them from scratch (Sojer, 2011). Software reuse is a sub domain of general knowledge reuse, and code reuse is a special case, yet one of the most important forms of software reuse (Sojer, 2011).

Given that there exists extensive research and resources on how to work with formative assessment of learners, but not directly related to programming, I wish to find out more about formative assessment connected to aspects of programming didactics. Based on the background provided from the interview study on assessment in

programming, where teachers elucidated about the use of external resources and particularly reuse of code and existing solutions, together with the coming learning goals considering code reuse, I find it appropriate that this is the aspect to be researched. Code reuse is a common way of developing software, and a huge field of research. At the same time, programming is becoming more and more common in education all around the world. In this project I wish to find out more about formative assessment in programming education in relation to code reuse.

1.2 Problem Statement

As described above, with the coming curriculum in Norway's primary educational system and IT as a growing academic discipline both in Norway and the rest of the world, there is a need for more research on IT didactics. Formative assessment is an important topic in didactics to research because it has a great impact on learning, while assessment is a widely debated, and teachers find assessment difficult. Code reuse is an important topic in programming, and in the coming Norwegian curriculum there are explicit goals for the learners regarding code reuse. In my research method course study, in interviews about assessment in programming, teachers noted that code reuse is both a challenging and interesting topic.

My literature research on programming, programming education, assessment and code reuse has not revealed any relevant literature on code reuse and formative assessment combined. There exists extensive literature on both topics separately, but to my knowledge there is limited research on them together. To the best of my knowledge, there is also limited research about formative assessment and code reuse in secondary education. Therefore, I consider gathering information on, and analyze, teachers experiences, reflections and knowledge regarding these topics in an exploratory manner a good place to start.

Given this background, I wish to research formative assessment and code reuse with the following problem statement:

What are teachers' experiences with formative assessment with regard to code reuse in programming education?

To answer the problem statement, I have stated three research questions. The first concerns teachers general attitude and awareness considering code reuse when they teach programming. The second concerns the challenges the teachers might have with formative assessment given that code reuse is a practice the learners frequently use. The third is to identify if there are any best practices, guidelines or other formative practice that can be done to overcome these challenges. The research questions are:

RQ1: What are teachers attitudes towards code reuse in programming education, and to how is it implemented in class?

RQ2: What are the challenges for the teacher when working with formative assessment regarding code reuse as a practice when programming?

RQ3: What are some best practices for conducting formative assessment from a teacher's perspective, regarding code reuse as a practice when programming?

I have found that teachers attitudes and approaches to code reuse varies from very positive and with an explicit focus on teaching it, to more restrictive and trying to avoid it in a learning setting. I have identified several challenges with formative assessment and code reuse, most revolving around the dilemma between learners using code reuse to be efficient and create complex programs, but at the expense of properly learning and understanding everything they do, and practicing general and specific programming skills. Finally, I have provided some conceptual and specific practises according to teachers' feedback to work with, overcome or mitigate these challenges. This includes formative assessment of various parts of programming, assessment of code reuse aspects in particular and integrating formative assessment continuously in the teaching practice.

1.3 Report Structure

In the following parts of the report, there is a background chapter further elaborating the theme for the project and its background. Then there is a chapter with theoretical framework defining and elaborating didactic concepts, formative assessment and code reuse in order to understand and being able to discuss the phenomena and data arising. Following there is a method chapter and a data collection chapter, describing methodological approaches, design and how the data was collected, followed up by a quality and ethics chapter discussing the ethics, validity, reliability, strengths and weaknesses of the project. Next comes a description of the analysis, followed up by results presented, and then a discussion part discussing the results. The last chapter of the report is conclusions, summing the content up briefly.

1.4 Contribution

The project provides two types of contributions. The first one can be seen as scientific or academic, confirming that code reuse in programming education is in fact a topic worth researching, and that there are connections between code reuse and formative assessment in programming education.

The second is of a more practical matter, identifying challenges for teachers and offering guidelines as to how to overcome these challenges.

2 Background

2.1 The Norwegian Context

Even though the project and many of its results can be generalized, the problem statement, theoretical framework and data collection is done in a Norwegian teaching context with a Norwegian curriculum, and this context should be clear and considered in order to understand the results. Nevertheless, research on programming education, code reuse and formative assessment is applicable to contexts outside the Norwegian secondary school.

2.1.1 Programming in Norwegian Primary and Secondary Education

Programming is not an entirely new concept in Norwegian primary education, but the interest and need for programming skills has drastically increased, together with the number of courses and learners learning programming. Already in the early 80's there were experiments with so called *electronic data processing* (from Norwegian "elektronisk databehandling") courses. However, it never gained much popularity, and programming was a special interest for a few (Sevik, 2016). In the early 2010's there have been several private and public initiatives to raise interest in technology and programming in Norway, such as *Lær kidsa koding* (Teach the kids coding), science centers, maker spaces, libraries and museums (Sevik, 2016). Starting in 2016, there has been a pilot project in Norwegian primary schools with programming formally as elective courses (Sevik, 2016) and at secondary schools as well (Utdanningsdirektoratet, 2016, 2017, n.d.-b). Starting autumn 2020, Norwegian schools execute a new curriculum *Fagfornyelsen*, where programming will be a part of several natural science courses, and in IT elective courses with renewed learning aims and content (Utdanningsdirektoratet, n.d.-a).

2.1.2 Code Reuse in the Norwegian curriculum

Even though code reuse is mostly regarded as a common practice when programming in this project, it is worth mentioning that there are explicit learning goals for the learners concerned with code reuse, and thus a motivation to study the phenomenon. In the current curriculum, there is a goal in the course *Information Technology 2* (13th grade), stating that the learner should be able to "create own, and use their own and others' functions or methods with parameters" (Utdanningsdirektoratet, n.d.-c, my translation). The goals for the new curriculum in 2020 are currently not decided, but the suggestions are as follows. For *Information technology 1* (12th grade), the learner should be able to "create and use own and others' functions with and without parameters and return values" (Utdanningsdirektoratet, 2020a, my translation). In the upcoming *Information technology 2* (13th grade) course, also not currently decided on, the goal considering code reuse states that the learner should be able to "generalize solutions, create reusable program code and utilize existing code in new programs" (Utdanningsdirektoratet, 2020a, my translation). There consequently seems to be an

acceptance and desire for code reuse as a programming practice, with an emphasis on systematically learning and utilizing the practice in the upcoming curriculum.

2.1.3 Formative Assessment Formally in Norwegian Educational Law and Curriculum

The Norwegian Directorate for Education and Training (UDIR) is responsible for the development of kindergarten, and primary and secondary education (Utdanningsdirektoratet, 2020b). Formative assessment has been one of the major areas of focus regarding development of teachers for UDIR. Norwegian learners have the right to proper assessment by Norwegian law (lovdata.no, 1998), and UDIR develops updated directives for how the laws should be interpreted and executed in practice. When publishing directives for the educational law in 2007 and 2009, UDIR based their directives on Norwegian experiences and international research documenting how formative assessment promotes learning and increases learning results, and directions from both years aim at an assessment practice that we can consider formative (Engh, 2017, p.18-19). Consequently, formative assessment is not only a good and interesting practice for research, but in a Norwegian context it should be considered law that teachers in some way or other conduct formative assessment as well as summative. See chapter 3.2 for elaboration on the terms. The directive clearly states that learners have the right to summative assessment, formative assessment and proper documentation of their education (lovdata.no, 2009).[1]

In many cases teachers need to use their skills and knowledge about their academic subject and the learners, in order to evaluate the quality of the academic achievement of the learner, and quality is very difficult to evaluate using a scale, standards or other quantitative tools (Engh, 2017, p.17). The Norwegian curriculum is *goal-oriented*, and one of the intentions behind this choice is that teachers' skills and results of the education can be evaluated (Engh, 2017, p.24). However, Norway differs from many other western countries because the specific goals for every subject are formulated qualitatively, meaning they cannot easily be measured. The learners' performance related to a goal cannot be evaluated in terms of the learners achieving a goal or not, but in the Norwegian context a *level of goal achievement* is evaluated for the learner, based on *characteristics* of goal achievement (Engh, 2017, p.24). An example from the curriculum in information technology states that, the learner should be able to "define variables and choose appropriate data types" (Utdanningsdirektoratet, n.d.-c, my translation from Norwegian). The goal does not state *to which extent* or *how good* variables should be defined, or *how* appropriate data types to be chosen. This makes the goals open to interpretations and also allows for its possibility to change and develop as the subject develops. This makes the base for Norwegian teachers somewhat unique, and might be important to keep in mind in terms of how assessment is conducted.

[1] For Norwegian readers, the terms "underveisvurdering", "vurdering for læring" and "formative assessment" are used interchangeably here, referring to Engh evaluating the terms as equal (Engh, 2017, p.29)

2.2 Pilot project

In the autumn of 2020, I conducted a research project in a research method course at NTNU, which has helped forming and understanding the research domain and questions asked in this project. It was an explorative interview research study asking teachers about their experiences with assessment in programming education in general, and when comparing them to other academic subjects they were teaching. Only asking about assessment in programming in general, multiple results indicated that code reuse would be an interesting topic to research further. The respondent teachers said that reusing existing solutions is very common among learners, and that they are encouraged to reuse. They explained how in programming, more than other subjects, the learners are able to utilize resources online, both for smaller inquiries and big, complex solutions to problems. They explained that this affected teaching in the way that reuse itself needed to be practiced, and that tasks and exercises to be assessed needed to be formed in such a way that code reuse could be utilized without copying full solutions. They also described having a clear understanding of how much the learners understood of their own reused work could be difficult, and that this was something they needed to work explicitly with. These results inspired me to look further into code reuse and assessment, and were important for the research domain, problem statement and research questions asked.

3 Theoretical Framework

In this chapter I will present the theoretical framework that will be used to report, describe, explain and discuss formative assessment and code reuse. Since this is an exploratory grounded theory study, the theoretical framework is not meant to be used to analyse collected data, but to discuss the phenomena, know how to collect data about it and how questions are asked. See chapters 4 and 5 for full method and data collection descriptions.

3.1 Didactic Dimensions and Relations

Assessment cannot be understood isolated from other didactic activities. In Norwegian didactics theory, this fact is illustrated with *the didactic relational model*, which emphasizes how different activities and elements of teaching are deeply interconnected. (Engelsen, 2006b). The model consists of

1. The learner
2. Goals
3. External factors
4. Ways of working
5. Content
6. Assessment

(Engelsen, 2006b, p.45-48) The model is widely used by Norwegian teachers both to plan teaching, and also to reflect on classroom activities. The important message is that when planning for and reflecting on, e.g. assessment, this is affected by the learner's current skills and knowledge, about how goals are set and understood, external factors like how many learners there are in a classroom or which programming language are used, what the content of exercises are and how the learners work with them. This will be important for analyzing teacher experiences not only considering clear assessment situations, but also considering factors that affect the assessment. The idea that different aspects of teaching are connected is an idea that can be generalized to all teachers. For a proper understanding of formative assessment, formative teaching activities, programming and code reuse, the different aspects need to be considered as a whole. The figure on the following page illustrates the didactic relational model and how different aspects of teaching are interconnected.

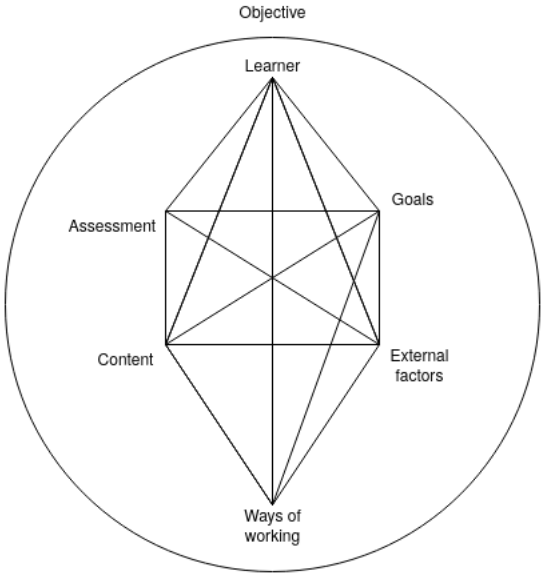


Figure 1: Didactic Relational Model

(Engelsen, 2006a, p.47, my translation)

Assessment should not be understood as an activity practiced outside of teaching, but a way of thinking integration between assessment and teaching, and the term *formative* can be used about the teachers teaching practice that is changed, modified or *formed* as a consequence of the assessment work (Engh, 2017, p.28). This is relevant when discussing the results, because assessment will not only be considered the assessment per se, but the teaching as a whole.

3.2 Formative Assessment

In this report, formative assessment refers to *assessment practice with the intention of creating positive changes for the further learning.* (Engh, 2017; Black et al., 2004, p.19). In practice, at a high level, it can be broken down and understood as a three-step process consisting of

1. Gather and understand information, and establish where the learners are in their learning
2. Decide and communicate desired goals, and establish where the learners are going
3. Establish what needs to be done to close the gap between the two former and get the learners there

(Black & William, 1998; Black & Wiliam, 2009b; Engh, 2017).

With the clear intention of creating positive changes in the learning of an individual learner, formative assessment differs from other types of assessment. An example is *normative assessment*, also known as group related or relative assessment, which is used to compare learners with each other, and assessment can be based on how other learners in a class or at a school perform (Engh, 2017, p.25). Another, which is an important contrast because it is a widely used and formally accepted practice, is *summative assessment*, which is assessment practice with the sole purpose of stating a learner's current knowledge, what the learner has achieved so far (Engh, 2017, p.28). Black et. al summarize that formative assessment "differs from assessment designed primarily to serve the purpose of accountability, or of ranking, or of certifying competence" (Black et al., 2004, p.10).

There is extensive research indicating that formative assessment has a great impact on learning. As mentioned in chapter The Norwegian context, the Norwegian Directorate for Education and Training base their commitment and priorities regarding formative assessment as not only a recommended practice, but also required practice by law, on international research on formative assessment (Engh, 2017, p.18-19). Professors Paul Black and Dylan William worked at the University of London, and Black belonged to a group of researchers from Great Britain called *Assessment Reform Group* (Engh, 2017, p.19). In their articles *Inside the Black Box* (Black, 1998) based on research reported in the article *Assessment and Classroom Learning* (Black & William, 1998), and *Working Inside the black box* (Black et al., 2004), they report a significant amount of research documenting that learners perform better and have higher learning achievements when teachers are taught how to, and perform, formative assessment. (Black, 1998; Black & William, 1998; Black et al., 2004; Engh, 2017). Black and William claim that educational standards evidently raise when improving formative assessment (Black, 1998, p.3).

Formative assessment is deeply connected with general formative activities in the classroom. Black and William explain that assessment as a general term is about activities providing information about the teaching or learning, and that the assessment becomes *formative* when this information is used to adopt the teaching work to meet the learning needs of the learners (Black & William, 1998, p.2). Extending the understanding of the term formative assessment being interconnected with other learning activities as mentioned in chapter 3.1 about didactic relations, it is relevant to speak of formative activities in general in the classroom. An example would be deciding how a project in a class should be worked with or reported, with the intention of being easy and result in fair grading, or with the intention of maximizing learning achievement for every individual learner. This may not be the same thing, even mutually exclusive in some cases, and we cannot consider formative assessment isolated from other formative learning activities. Black and William explain that

> Practice in a classroom is formative to the extent that evidence about student achievement is elicited, interpreted, and used by teachers, learners, or their peers, to make decisions about the next steps in instruction that

> are likely to be better, or better founded, than the decisions they would have taken in the absence of the evidence that was elicited.
> (Black & Wiliam, 2009b, p.9).

3.2.1 Formative Practice

Having considered formative practice and activities is relevant to understand formative assessment, and to evaluate and discuss the results of the research study presented in chapter 8. These are some of the formative assessment practices studies show have the greatest impact on learning. The activities, or practices are

1. Clarifying and sharing learning intentions and criteria for success
2. Engineering effective classroom discussions
3. Providing feedback that moves learners forward
4. Activating learners as owners of their own learning, and instructional resources for one another
5. Engineering learning tasks and exercises that elicit evidence of learner understanding

(Black & William, 1998; Black et al., 2004; Black & Wiliam, 2009b).
The different activities are not isolated from one another, and must be considered interconnected. For instance, engineering effective classroom discussions can be a way of clarifying and sharing criteria for success. The practices are enumerated and formulated slightly different in different literature, but the five aspects presented here preserve the content.

Clarifying and sharing learning intentions and criteria for success. Even with motivated learners, and good, educational learning activities, the learners need to have a clear understanding about their learning goals, the intentions behind them and the success criteria related to the goals. In terms of programming and code reuse it becomes important for learners to know and understand what the exact goals of e.g. an exercise is, and whether or not they succeed with programming everything manually or they can reuse solutions and still meet the criteria. This practice is especially relevant in the Norwegian context where the general learning goals are formulated qualitatively as described in chapter 2.1, and the teacher needs to be very clear about what is quality in code and programs. Formative practice related to this is the formative use of summative tests, where the teacher can help the learners understand the goals and success criteria not only through grading, but discussing and commenting tests with the learners (Black, 1998, p.6). Also, learners should learn and perform self-assessment, so they can better understand the purposes of their learning and have an understanding of what they are supposed to achieve (Black, 1998, p.6).

Engineering effective classroom discussions. A significant amount of the assessment happens in the daily work including talking and discussing with the learners

individually, in groups or in the whole classroom. The dialogue between the teacher and the learners should be thoughtful, reflective and focused on evoking and exploring understanding (Black, 1998, p.8). Learners should have the opportunity to express their understanding to initiate interaction for aiding learning (Black, 1998, p.7). Teacher questioning is a common part of classroom dialogue. Questions should explore issues that are critical to the development of the learners' understanding, there should be considerable wait time to let the learners think, all answers, right or wrong, should be used to develop understanding, and follow-up activities should create opportunities to extend the learners' understanding (Black, 1998, p.7). The teacher's role in classroom discussions and questioning is to lead an exploration and development of ideas, where all learners are involved and active. (Black et al., 2004, p.13).

Providing feedback that moves learners forward. As learners gain an understanding of their current skill of knowledge level feedback needs to be provided in such a way that the learners know how to achieve their goals. Feedback should be about particular qualities of the learner's work, include advice on what the learner can do to improve, avoid comparison with other learners or summative judgement, and should cause thinking to take place (Black, 1998; Black et al., 2004). As mentioned for questioning, learner activities such as answering questions or providing written tasks should encourage learners to develop and demonstrate understanding (Black et al., 2004, p.14). An effective formative practise in terms of feedback is comment marking. Comments should identify what is done well, what needs improvement, give advice on how to achieve improvement, and the learner should have active involvement in the process, having the opportunity to respond to comments, set questions and mark answers (Black et al., 2004, p.13). They should also practice peer- and self-assessment through giving each other feedback to better understand how their work can improve (Black et al., 2004, p.14).

Activating learners as owners of their own learning, and instructional resources for one another. The aims of the learners' work and what needs to be done, including success criteria for learning achievements should be made transparent to the learners. Further, learners should be taught habits and skills of collaboration in peer-assessment, and they should be encouraged to assess their own and each other's progress toward achieving their learning goals (Black et al., 2004, p.20). Peer- and self-assessment make distinct contributions to learning, and Black et. al claims that they "secure aims that cannot be achieved any other way" (Black et al., 2004, p.15). Self- and peer-assessment is an essential component of formative assessment, and the three steps presented earlier consisting of present position, desired goals and means to close the gap between the two, must be understood also by the learner before they can take action to improve their learning (Black, 1998, p.6).

Engineering learning tasks and exercises that elicit evidence of learner understanding. Exercises for tests, work in class and homework is an essential part of learning and practicing skills, but should be done in a formative manner. Exercises should be clear, relevant and justified in terms of the learning goals (Black, 1998, p.7). Fur-

ther, opportunities for the learners to communicate and take part in their progress should be elicited, including discussions, communication about activities and marking (Black, 1998, p.6). The feedback given to the learners should guide them to improve, and the learners should be given the opportunity to improve the assessed work or exercises (Black, 1998).

Even though interconnected, these activities seek to achieve appropriate formative assessment, and are not considered as "every formative activity" in the classroom. For instance, it could be argued that engineering exciting, motivating and interesting exercises, because motivation promotes learning, can be seen as formative. This is an important aspect of code reuse as a practice because it allows learners to work with exactly more motivating and interesting problems, perhaps outside their skill level. However, this is not elicited further in this project, even though an important aspect of code reuse in programming education.

Considering the definitions, descriptions and examples, it is clear that formative assessment as a practice does not only consider the teacher as an agent performing this practice, but also the learners, including the interactions between teacher and learners, and among the learners themselves. The teacher is responsible for the learning environment, while the learner is responsible for the learning (Black & Wiliam, 2009b, p.4). In the research project and this report, the focus is on *the teacher, their experiences, challenges and practice*. However, assessment is performed in the interactions between the teacher, the learners and their peers, and supplementary research should consider the learner's perspective as well. The following image illustrates the different aspects of formative assessment, and the roles of the teacher, the learners and their peers.

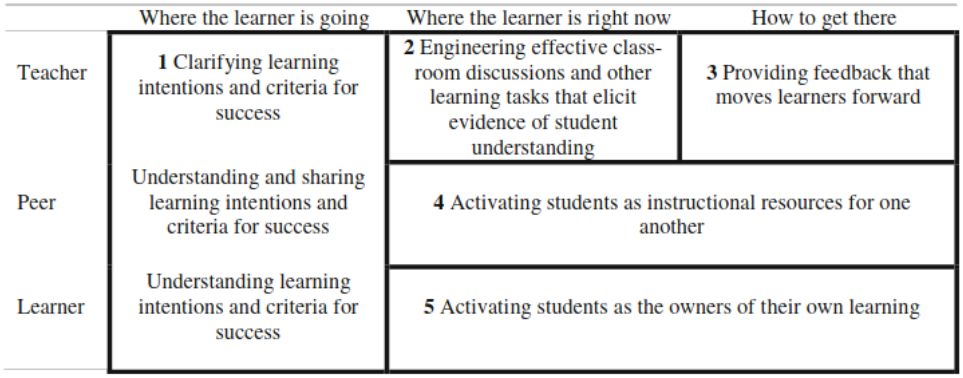


Figure 2: Aspects of Formative Assessment

(Black & Wiliam, 2009a, p.8)

Though extensively researched, formative assessment is not a concept or teaching practice based on a general or grand theory. It can be thought of as an idea or principle for teaching and education, and will be conducted differently for different

teachers, learners, classes, schools, countries and especially academic subjects. It will be in constant development, and teachers will always find their own practice (Engh, 2017, p.28). That makes studying formative assessment in programming, framed around code reuse as a phenomena, particularly interesting. Teachers can follow guidelines and principles, but formative assessment cannot be conducted following a recipe (Engh, 2017).

3.3 Code reuse

I will here define what code reuse is, and explain the process of code reuse when programming.

Software reuse is the process of using existing software or software knowledge to create or develop new software, rather than creating it from scratch (Frakes & Kang, 2005; Sojer, 2011). Code is the most commonly reused artifact, while software reuse in general deals with the reuse of designs and design patterns, architectures, cost estimates, project plans, requirement specifications, test cases, user interfaces, documentations, customized tools, methods and business models among others (Frakes & Kang, 2005; Sojer, 2011). Consequently, both the terms software reuse and code reuse will appear in the report, but the project is about code reuse, which is a special though common type of software reuse.

In this project, code reuse is *the process of using existing code or code knowledge to create or develop new code, software or programs, rather than creating them from scratch*. Code reuse can include, but is not limited to, reusing lines of code directly, code structures to be modified, functions, libraries and components.

Software reuse and code reuse are very important topics both for working with programming and software development, and as a field of research. There is extensive research about reuse, and even an international conference every year devoted to software reuse with research presented, called International Conference on Software and Systems Reuse (*ICSR*, n.d.). Most software systems are not entirely new, and reusing code is a widely practiced in order to take advantage of existing resources when developing software and programming (Agresti, 2011; Frakes & Kang, 2005). Studies show that developers often start programming by searching the web for existing solutions for their current task, and that reuse is a common motivation for code searches on the web (Umarji & Sim, 2013; Ayala, Franch, Conradi, Li, & Cruzes, 2013). Nakakoji et. al goes as far as claiming that everything is a remix, and that "programming is now viewed as basically remixing" (Nakakoji, Yamamoto, & Nishinaka, 2013, p.17). This is of course slightly exaggerated, but gives an indication about how common and important this practice is.

For businesses, there are good incentives to reuse code and software besides the fact that it already exists and that developers do not want to "re-invent the wheel". Some of the main reasons software developers and businesses systematically reuse software and code are: improving development efficiency, productivity and increase development tempo so that systems can be delivered on time, with lower costs and

improved quality and system reliability. This is achieved as reused code is often well tested, debugged and documented, which makes it easier to develop bigger and more complex, yet maintainable systems (Agresti, 2011; Frakes & Kang, 2005; Haefliger, von Krogh, & Spaeth, 2008; Sojer, 2011).

Another aspect of code reuse, which is also very important in terms of learners learning programming, is that the reuse allows leveraging expertise. This means that developers can specialize on certain areas and develop reusable code that can be used by other developers, in this case the learners, who are not experts, but still need to use the code with this functionality (Sojer, 2011). For learners, this means that even with limited programming skills, they can do relatively more complex, interesting and motivating projects using reusable code where their skills come short.

The process of code reuse consists of

1. Retrieving existing code artifacts
2. Understanding and evaluating them
3. Modifying them to fit the new context
4. Integrating them into the new code, program or software

(Sojer, 2011)
I will here explain and exemplify the terminology, and describe characteristics of the steps in the process of code reuse presented above.

Reusing lines of code or code structures is also called *snippet reuse*, and covers "reusing lines of code directly" and "code structures to be modified" from above. Multiple continuous lines of code, or the structure of a larger block of code, is being "scavenged" from existing programs and used in the programmers own program (Sojer, 2011). The snippet can be directly copied and reused, or details can be changed or deleted while the structure is retained (Sojer, 2011). Reusing functions is also considered as snippet reuse for relatively small or basic functions, and component reuse as the complexity grows.

Reusing code artifacts designed explicitly to be reused is also called *component reuse*. The component are usually larger blocks of code or longer programs than what we consider snippets, and can be considered as encapsulated code knowledge such as complete algorithms, data structures or programs (Sojer, 2011). In commercial software development, the idea of reusing components is using code that has been developed, documented, tested and sometimes even certified, saving the developer the work (Sojer, 2011).

A library typically consists of a repository with reusable code, a search interface for finding the code to be reused, a representation method, and facilities for changing the code to fit a certain task and quality assessment (Frakes & Kang, 2005).

Retrieving existing code artifacts. Retrieving code artifacts means searching for, and finding, the code to be reused in a program. An important part of retrieving, or more specifically *finding*, existing code artifacts, is searching the web. Studies show

that searching the web is one of the most common activities for software engineers, and code reuse is one of the main motivations for code search on the web (Sim & Gallardo-Valencia, 2013). When finding snippets, the programmer needs to consider where and in which parts of other software reusable parts exist, while retrieving components can be more straight-forward as they are often stored i libraries and categorized according to functionality (Sojer, 2011).

Understanding and evaluating retrieved code artifacts. After the code to be reused has been retrieved or found, the programmer needs to understand and evaluate it. Because components typically are designed to be reused, this part is easy for components and the programmer can look at the documentation, predefined interfaces etc., or trust its reputation by peers, to evaluate it (Sojer, 2011). For snippet reuse, the programmer typically needs to look through the code line by line, look for reusable parts, and evaluate it (Sojer, 2011). Ayala et. al breaks the selection of a reusable components into *identification of candidate components*, where the developer locates and acquires information about the reusable artifact, *evaluatng components with respect to the expected requirements*, to assess if the artifact covers system requirements, and *choosing suitable component alternative(s)*, where the developer compares different candidate artifacts to choose one(s) that fits the requirements in the most appropriate way (Ayala et al., 2013).

Modifying existing code artifacts to fit the new context. When suitable code to be reused is retrieved and evaluated, it may need to be modified and shaped to fit the programmer's specific task. Both snippets and components might need to be modified to fit certain data types, versions of programs or types of tasks, and the code needs to be changed manually (Sojer, 2011). For many types of component reuse, however, developers can modify the component through parameters, because the original developer of the component has predicted that the component might need to be slightly modified and has provided alternatives to fit different situations (Sojer, 2011).

Integrating existing code artifacts into the new code, program or software. Integrating the code is similar to modifying, but includes the actual integration and execution of the reusable code in the new. Typical tasks would be changing variable names, comments and documentation (Sojer, 2011). For snippets, an understanding of every detail is typically required, while for components most software development environments allows integrating the new code easily (Sojer, 2011).

Though emphasizing positive effects of code reuse such as increased quality, more complex systems and efficient programming, there are also quality risks associated with the practice. If the person programming does not understand the code being reused, and for efficient reuse they are not always required to, the code may impact the complete program or software negatively (Sojer, 2011). Especially when we move our focus to learners *learning* programming while practicing code reuse, there might be complications and challenges, which this project is partly trying to shed light on. Code reuse in its essence is not really about the reuse of code per se, but the reuse of human problem-solving (Haefliger et al., 2008), and for learners, code reuse

can be a way of programming like developers do "in real life", but also rob them of opportunities to solve problems and learn.

There are also some interesting learning gains in web searches, code reuse and copying. Gallardo-Valencia and Sim observed 24 developers at 3 software companies in a study, and found that developers searched the web to remember programming details they had forgotten, clarification for a high-level understanding of something they wanted to implement, learning to acquire new concepts, and the need for a tool or open source project/component, i.e. software reuse (Gallardo-Valencia & Sim, 2013). They also found that as much as 82% of web searches were ad hoc done to remember syntax details, clarify implementation details or fix bugs, and learn new concepts (Gallardo-Valencia & Sim, 2013). Recent studies on teaching computer science also suggests that the act of copying and mimicry can have good impacts on learning (Zander et al., 2019). I will discuss this in the discussion chapter.

4 Methodological approach

When performing research, there are several choices to be made and communicated clearly as to what the researcher considers true knowledge, or rather, what is considered empirical knowledge in the specific project, if the knowledge can be considered objective and whether it is generalizable or not.

In this chapter I will explain how the project relates to qualitative research with a phenomenological approach and a grounded theory design, and the nature of qualitative interviews. In the following chapter I will go into further details concerning the consequences this approach has for the data collection.

4.1 Qualitative Research

The design of a research study begins with the background information, problem statement and how the questions are asked (Robson & McCartan, 2016, p.72). In the introductory chapter, I argue that there is not much empirical research nor theories available that consider the particular topic of formative assessment of learners' code reuse, and that it is therefore important to explore if there are any special challenges and practices revolving around this. Given that the answers to these kinds of questions cannot be easily quantified, and are explorative in nature, I am methodologically positioning at a *qualitative* research design. Some typical features of qualitative research are that data is often in a non-numerical form, a focus on meanings, context and situations are described by those involved, and objectivity and generalizability are less valued because the context, the researcher and the social situation is prioritized (Robson & McCartan, 2016, p.20). A quantitative approach would seek to obtain numerical data with statistical analysis, focus on measurement and quantification, and objectivity and generalizations of results is sought and prioritized (Robson & McCartan, 2016, p.19). For a quantitative approach regarding this project, the research questions would perhaps consider *how much* code reuse is used as a practice, *how many teachers* experience challenges related to code reuse, etc. Given that we do not know much about these challenges yet, I find it appropriate to attempt to identify, describe and understand these with a qualitative approach, to acquire insights into teachers' experiences and practice regarding this.

In the research project, I am interested in understanding challenges and practice regarding formative assessment and code reuse from the teachers' own perspectives, and describe the phenomena the way they experience, sense and understand them. This is called a *phenomenological* approach, and focuses on understanding how humans view themselves and the world as they experience it (Robson & McCartan, 2016; Kvale & Brinkmann, 2015). It is an understanding of the world considering that knowledge and reality is that which humans perceive or apprehend (Kvale & Brinkmann, 2015, p.45). The research is influenced both by the teachers whom I try to understand, and my interpretations. As Robson and McCartan put it, "The research methodology informed by what is often called 'interpretive phenomenology',

seeks to reveal and convey deep insight and understanding of the concealed meaning of everyday life experiences" (Robson & McCartan, 2016, p.165).

By considering teachers' experiences, reflections, expertise and knowledge, together with my questions and interpretations, I am taking on a *constructivist* view of what is considered empirical knowledge. This means that knowledge, social properties and meaning are constructed through interactions and interpratations between humans, rather than existing in its own right and having a separate objective existence (Robson & McCartan, 2016, p.24).

Choosing to perform qualitative research with a phenomenological approach and constructivist view of empirical knowledge, has some consequences for the research project. Investigating if there is something interesting to discover concerning formative assessment of learners and code reuse, is narrowed down to asking what teachers' experiences with the phenomena are, particularly in terms of their *challenges* and their *practice* in overcoming these challenges. This has some important implications for the possibilities and restrictions of the research. First, *challenges* and *practice* are what the teachers consider them to be. I am not defining that a challenge is, for instance, when assessment is done more slowly than usual or "stops" in some kind of way. The research project is explorative is nature, and knowledge and data is produced during the research in the interaction between me as the researche and the teachers. Second, the answers to the questions cannot answer *how many teachers* experience these challenges and use these practices, or other generalizable answers. The research can identify, describe and gain insight to what some programming teachers consider challenging, and what some programming teachers consider good practice. This has intrinsic value and can be learned from and used to form non-definitive guidelines, and they can point further research in more specific directions. Third, I as the researcher become important for the data collection and the interpretations of the data, and the results cannot be viewed separate from me, my biases, knowledge of the topics, social skills and reliability as a researcher. I will discuss this further in chapter 6 which concerns the quality of the research project.

4.2 Grounded theory

Grounded theory is a research design aimed at developing theory from data collected, as a contrast to using existing theoretical framework to collect and analyse data. The research design is applicable to a wide variety of phenomena, and particularly useful when there is a lack of theory and concepts to describe the phenomena, or the theoretical approach to be used is not clear or is non-existent (Robson & McCartan, 2016, p.80,161). The aim is try to generate theory from the data collected during the study, which relates to the situation forming the focus of the study (Robson & McCartan, 2016, p.150). Grounded theory is not a theory itself; the theory is derived from the study in particular, and "grounded" in the data obtained during the study (Robson & McCartan, 2016, p.161). It is a strategy for doing research and a style for analysing the data obtained from the research, and both have particular procedures

and techniques (Robson & McCartan, 2016, p.161). The data collection, analysis and theory development are thus interspersed throughout the study (Robson & McCartan, 2016, p.80).

Choosing to decide on grounded theory as research design has some important implications for the project. First, it builds on the problem statement, research questions and the lack of knowledge about the meeting between formative assessment of learners and code reuse when learning programming. Second, it means that there is not a theoretical framework provided in order to analyze the data collected; the aim is to derive theory about teachers' challenges and practice, from the data itself. There is of course, as seen in chapter 3, theoretical frameworks to talk about code reuse and formative assessment, but the theory derived here aims to explain the particular meeting between the two. It is nearly impossible to conduct a research study without any pre-existing theoretical ideas or assumptions (Robson & McCartan, 2016, p.162). Third, it affects how the data is collected. In grounded theory research, interviews are the most common data collection method (Robson & McCartan, 2016, p.162), and in the next sub section I will give an account of qualitative interviews, before moving on to the data collection and analysis.

4.3 Qualitative Interview

As my research questions revolve around teachers' experiences, thoughts, reflections and expertise regarding formative assessment and code reuse, and given that I am going for a phenomenological approach with grounded theory design, qualitative interviews makes sense for data collection. Qualitative research interviews aim at understanding the world from the respondent's point of view, and shed light on experiences and perceptions (Kvale & Brinkmann, 2015, p.20). Doing qualitative interviews fits well with a phenomenological approach, because they give access to humans' fundamental understanding of their world, and their experiences form the foundations of the more abstract, scientific theories about the phenomena (Kvale & Brinkmann, 2015, p.47). Qualitative interviews are commonly used with grounded theory design, and are open for personal experiences and descriptions separated from pre-known theories, and the interviews themselves are a specific context in which knowledge is produced (Kvale & Brinkmann, 2015, p.76-77).

5 Data collection

In the following sections, I will describe how the data was collected. In qualitative research such as this project, the researcher becomes more important because the data collected and how it is interpreted is very dependent on the researcher's interactions with the respondents, knowledge and expertise. Therefore, I have tried to explain with as much detail as what is practical the whole process of the data collection. See also chapter 6.2 for information concerning research quality with more discussion.

5.1 Personal interview

Given the methodological approach described in the former chapter, I utilized semi-structured, personal interviews. Personal interviews are used when trying to understand topics from the respondent's own perspectives, and seeks to obtain descriptions, experiences and reflections - and especially interpretations of the described phenomena (Kvale & Brinkmann, 2015, p.46). It is semi-structured in the sense that it is neither an open conversation, nor a strictly framed Q&A with little options (Kvale & Brinkmann, 2015, p.46). In this way I could be specific with respect to which topics I wanted to talk about, e.g. code reuse and formative assessment instead of general software reuse or general assessment, and at the same time be open to new aspects, experiences and descriptions which could be used to develop the theory and the interviews further. Semi-structured interviews are performed using an interview guide that narrows the interview down to certain topics and can contain suggestions as to which questions to ask (Kvale & Brinkmann, 2015, p.46).

5.2 Respondents

Sampling in grounded theory studies is purposive, meaning I was not seeking a representative sample just for the sake of it (Robson & McCartan, 2016, p.163). I chose to interview teachers who worked at high schools, and had experience with teaching programming, as expert interviews in the field of programming education. In grounded theory, this type of sampling is sometimes also referred to as *theoretical sampling*, meaning the respondents are chosen in order to help the researcher to formulate theory (Robson & McCartan, 2016, p.163).

I will briefly summarize the attributes of the respondents here and further discuss them. I ended up conducting interviews with twelve persons. Amongst the respondents there were men and women, who had been teaching programming in some way for anything from two to over thirty years. They were all teaching at different schools, geographically from five different counties in Norway, and teaching different programming subjects at high school level. As an exception, one of the teachers indeed worked at a high school, but had been teaching programming at secondary school level. However, I have taken this into consideration with regards to the analysis if comparing statements, experiences etc.

Regarding the amount of respondents, this varies a lot depending on the purpose of the data collection, varying from just a few respondents to interviewing an increasing number of people until the data collection is "saturated", meaning nothing new is added to the data at some point (Kvale & Brinkmann, 2015; Robson & McCartan, 2016). My research questions concern teachers' experiences with regards to challenges and practices, and thus does not have the need to be saturated in the same way as if I would ask for *every* challenge and practice. However, I wanted to conduct as many interviews as possible to be able to discuss significant similarities and differences, and have both a broad collection of data for different results, and being able to assert claims about the topics if possible. All in all, my goal was to conduct as many interviews as possible, given the scope of the project, and i ended up with interviewing 12 teachers for the project.

I contacted the respondents by sending e-mails to different high schools which I had previously identified as schools offering programming courses. The initial e-mail had limited information about the project, but summarized it, and if interested, the teachers received full information about the project and the interviews, so that they could make an informed decision on whether they wanted to participate or not. Out of thirteen teachers who received full information, twelve said yes to participate in an interview. See chapter 6.1.2 for more about the information and consent to participate.

5.3 Information Preparing teachers

Amongst the information to the respondents about the purpose of the project, data processing, etc., I had also added a note with some basic definitions and examples of formative assessment and code reuse. Doing this was quite the dilemma, because defining the terms and providing examples before the interview situation, is definitely leading and hurts the objectivity and personal perception of the topic from the teachers. However, formative assessment is an extensive and complex topic, and together with code reuse, I did not want the respondents to be confused about the project. Therefore, I tried to provide as basic definitions as possible, together with very general examples. I think this was an appropriate decision, especially because the exact term *code reuse* is not used in the current Norwegian curriculum (however, seems to be used in the future), and because *formative practice* is not specific things teachers do, but an idea or concept of a practice. Put shortly, I expected the topics combined to be too confusing to not provide some information beforehand.

5.4 Conducting the Interviews

The twelve interviews were conducted during the weeks between 21.04.2020 and 04.05.2020. The aim was to spread the interviews out as much as possible. I had at most three interviews in one day, and at least one hour break between each interview. The interview, with the voice of the respondent, their face, body language, their mood and tone give a richer access to the respondents opinions than only the

later transcriptions, so taking breaks between the interviews is important to reflect on what has been said and which results may have come out of the interview (Kvale & Brinkmann, 2015, p.161). These spontaneous impressions can be very valuable for later analysis of the written transcriptions (Kvale & Brinkmann, 2015, p.161).

There are several ways of conducting interviews considering later documentation and analysis, including voice recording, video recording, written notes and by memory (Kvale & Brinkmann, 2015, p.205). I chose to use voice recording for the later analysis, and written notes during the interview for follow-up questions and topics that would arise. When recording sound, the researcher can concentrate on the interview's dynamics, choice of words, tone, pauses and other verbal attributes (Kvale & Brinkmann, 2015, p.205).

The interviews were done with computer video chat software, but the recordings with an external sound recorder not connected to the computer. See chapter 6.1.3 about data processing for details.

In the beginning of each interview, before formally starting to ask questions, I took some time to speak naturally and freely with the respondent to establish a relationship. This is important so the respondent gets an impression of the interviewer before speaking about their life, and contact and trust is established by the interviewer showing interest, understanding and respect by listening to what the respondent has to say (Kvale & Brinkmann, 2015, p.160).

The interview was introduced with a briefing, where I tried to define the situation, explain the purpose of the interviews, how I was going to record it and use the recordings, etc., and then asked if the respondent had any questions before we began.

In addition to focusing on the thematic questions, the interviewer should keep in mind the later analysis, verification and reporting of the interviews (Kvale & Brinkmann, 2015, p.165). During the interviews, I was focused on trying to verify that I had understood what the teachers were saying by asking clarifying questions where I was in doubt later. This was especially important in the parts of the interview where I first asked about general formative assessment when teaching programming, then about code reuse, because I didn't want to in the analysis claim that the teacher did some practice in relation to code reuse, if it indeed only was meant as a general statement or related to something different. Therefore, I frequently asked questions like "Do I understand you correctly, if you are saying...", and "Can you explain what you mean by that, and maybe provide an example?". Attempts to clarify the statements of the respondent will make the analysis more certain, and also show the respondent that the interviewer is indeed listening and trying to understand (Kvale & Brinkmann, 2015, p.83-84,165).

After the main questions I started finishing up the interview by asking some questions about whether the respondent collaborated with other teachers about the topics, and whether they had discussed code reuse explicitly. This was both to start finishing up, and to get some last reflections not only related to the teacher personally, because teaching is a collaborative matter done in schools as a whole, even though the teacher is alone in the classroom.

At the end of the interview, we did a debrief, and I asked the respondents if they wanted to add anything, if there was anything there was not room to talk about etc. Debriefing like this gives the respondent and opportunity to speak about topics they may have wanted to talked about or been worried about during the interview (Kvale & Brinkmann, 2015, p.161). I also continued to speak to the respondent for a while after stopping the recorder, and told them how they could stay in touch if they had anything they wanted to add later, or they wanted to pull back their participation.

I did three test interviews before conducting the real interviews with the teachers. Two of my colleagues had knowledge about my project, and I tested the interview process on them first to get a good feeling of the choice of words and the flow of the interview. The third one was new to the project, and we did a full scale test interview to fine-tune the questions, the build-ups and the interview as a whole. This gained some invaluable information about how I should ask the questions and conduct the interviews so that the respondents would understand the questions and topics to discuss.

5.4.1 Interview guide and Questions

In this section, I will describe the interview guide which I used, and discuss how it was used.

Prior to the interviews, I wrote an interview guide. An interview guide is a manuscript with a structure for the interview, and usually contains topics to be covered and questions to be asked (Kvale & Brinkmann, 2015, p.162). My interview guide consisted of certain topics to cover with suggestions to questions and follow-up questions, which is common for personal qualitative interviews (Kvale & Brinkmann, 2015, p.162). My guide consisted of an introductory part with formal information about the project and the interview, like a briefing, followed up with some generic questions about the teacher to get started. Then, a main part with questions related to the problem statement and research questions, and a closing part making sure I didn't miss anything and followed by a debrief of the interview situation. I will further explain and discuss the design of the questions.

Code reuse, especially in the isolated context of formative assessment, can be thought of as a quite particular topic. I first asked about formative assessment in general, and what they considered as code reuse in their classroom. The goal of the design of the interview structure further was to begin with general questions, then move on and investigate from what the respondents said and identified, and ask more specific questions if not all topics I wanted to discuss were covered. The general approach for each main topic would be to first ask about the respondents' thoughts on formative assessment when teaching programming, then after hearing how they work with formative assessment in general, getting some insights in their practice, etc., I would ask about code reuse. The reason for this slightly indirect approach is that asking for *challenges and practices regarding formative assessment of learners with regard to code reuse* is a very complex theme consisting of many different topics.

Interview questions should be short and concise (Kvale & Brinkmann, 2015, p.163).

For each question there were sub questions to elaborate. For semi-structured qualitative interviews, the interview guide keeps track of the topics to discuss, but the interviewer can choose in which order to ask the questions, if all the questions need to be asked, etc. (Kvale & Brinkmann, 2015, p.162). After a set of questions, I would do an evaluation on which topics we had spoken about and not, and then possibly move on with more specific topics regarding formative assessment.

I split the following questions in the three steps of formative assessment; information gathering, goals and decisions on what to do to bridge the gap between the two. I would again ask first in general terms of assessment, because I recognized that this is most common ground for teachers, and then ask more specific about the code reuse part. For instance, I would set the stage explaining that we were now to consider formative assessment as only gathering information about the learner, understanding the learner's skill level etc., and after some elaboration from the teacher, ask if there were any challenges with this work with regards to code reuse. This way of asking the questions gave interesting information about what the teachers considered formative assessment in general, which is important for the interpretation of their answers, and also helped staging the questions directly answering the problem statement more easily.

Summarized, the interview guide had topics, specific questions and specific follow-up questions, but with the possibility not to ask all the questions if not needed, and to go more deeply into the topics the respondents spoke about, which is common for the types of qualitative semi-structured interviews, in explorative project designs (Kvale & Brinkmann, 2015, p.46,162). There were both direct thematic questions regarding the theoretical perceptions of the topics, and dynamic questions asking about how the respondents experienced the topics, playing on the interpersonal relation in the interviews (Kvale & Brinkmann, 2015, p.163).

5.5 Transcription

In this chapter, I will describe and explain the transcription of the interviews. After the interviews were conducted and recorded, I transcribed the interviews. Transcribing is writing what has been said, with more or less details included, in order to be able to analyse it later (Kvale & Brinkmann, 2015, p.204,205). Transcribing means transforming, to change from one form to another, and a transcription is an artificial construction neither covering an oral conversation, nor a written text with formal requirements (Kvale & Brinkmann, 2015, p.205). The shape, the meaning and the context is changed, and several choices need to be made when transcribing an interview.

For the data in this project, there are several levels of abstraction. First, because the interviews were conducted over video and voice chat software, there is an abstraction from a physical meeting, to what you can see and how you can communicate using a camera and voice. Some of the body language in physical conversation will

get lost. Second, there is an abstraction from the webcam meeting to the recording, where all body language, including facial expressions are lost. The third abstraction is from the oral interview conversation to written form, where vocal pitch, intonation and respiration is lost. Transcriptions are reduced and decontextualised reproductions of conversations (Kvale & Brinkmann, 2015, p.205). Though the interviews were conducted, transcribed and analysed in Norwegian, quotes from the respondents are presented translated to English in the report, which is also a level of abstraction where meaning can get lost. To mitigate, though not fully overcome, these issues I took notes during each interview, and also tried to transcribe as fast as possible to be able to reproduce factors of the social context important for the analysis of what was being said in the recordings.

When it comes to rules or guidelines for transcriptions, there really are no specific rules to follow, and how the transcriptions should be conducted depends on the project (Kvale & Brinkmann, 2015, p.207,212). There are rather some standard choices to make, considering whether statements should be transcribed exactly as they were said, word for word, etc., and which dimensions of oral conversations should be included, such as pauses, intonation etc. (Kvale & Brinkmann, 2015, p.207,212). The choices depend on whether the transcriptions are used for e.g. a detailed language analysis, an analysis of a conversation or of the stories being told by the respondents (Kvale & Brinkmann, 2015, p.208). For my project I chose to transcribe the recordings word by word, trying to follow the flow of the conversation. If there were very specific intonations, I commented it in the transcriptions with explanations of how I interpreted the statements, and I also marked with "..." for short pauses, or with a comment for longer ones. For a research project where the way things are said are the focus of the research, elements like intonations and pauses would perhaps need to be marked more specifically (Kvale & Brinkmann, 2015, p.208). For my recordings, I tried to mark pauses relative to the general speed of the conversation with each respondent, and wrote comments where the respondents may have sounded (or was perceived by me) unsure or insecure. For instance, I would write a comment if there was a really long break before a respondent would say things like "No, I don't think there is a challenge...", or if they said it clearly sarcastically. However, there is always room for mistakes for these things, and it's important to consider single statements, questions and answers and longer elaborations in whole. Nevertheless, I didn't find it necessary to be more specific about language details than this. Including breaks, repetitions and tone is relevant for a psychological interpretation of the feelings of the respondent, or the meaning of a statement or claim (Kvale & Brinkmann, 2015, p.212). I assume the teacher's were honest given the nature of the questions not being very vulnerable, but one can never be a hundred percent sure.

As for the technical procedure, I first transcribed the interviews word by word with approximately half speed, taking breaks where needed, then listened to the whole interview with normal speed to fill in where I was not secure, and finally listening through the full interview again and reading the transcription, making sure everything was correct.

6 Quality and Ethics

In this chapter, I will describe, explain and discuss some of the ethical aspects considered for the project and especially the data collection, some formal requirements with regards to conducting research studies, and the quality of the project as a whole.

6.1 Ethics

When conducting a research project, especially involving humans, there are several ethical considerations to take into account. In every step of the project, from how I asked my research questions, to how I collected the data, and to how I have reported it in this report, I have followed guidelines given by *The National Committee for Research Ethics in the Social Sciences and the Humanities*, NESH (NESH, 2016). NESH is a Norwegian national organization that draws up guidelines for research ethics in the social sciences, the humanities, law and theology, and have good and thorough guidelines to follow (NESH, 2015). I will here and in the following sub sections report some of the most important ethical considerations regarding my project, however, it is not an exhaustive ethical evaluation of the project as a whole.

The most important ethical requirements to consider for this project, were those related to the respondents, i.e. the teachers. The respondents have the right to be treated with respect and dignity, to have their personal information and privacy secure, to confidentiality regarding their personal data, to be well informed about the project and how all data is going to be processed, and to consent (or not) to everything that is going to take place with them involved in the project (NESH, 2016). In addition to the individual teachers, I have also considered teachers as a group and to some extent schools as institutions, when I have considered vulnerable agents. It has been important not only to protect the respect and dignity of the teachers, but also to take into account that teachers as a group do not come off badly or be represented in a bad, undignified or disrespectful manner in the report. In the following sub sections I have described some of the main concerns and how I proceeded.

6.1.1 NSD

At NTNU, both professors and students who are to process personal data in a research project, need to do an evaluation on whether they need to report their project to the *Norwegian Centre for Research Data*, NSD (NTNU, n.d.-a). NSD is a Norwegian organization owned by the Ministry of Education and Research, and is a national archive and center for data research (NSD, n.d.-a). Because I for practical reasons was going to collect the names and e-mail addresses of the teachers to interview, combined with recording their spoken statements on a recorder during the interview, I was required to fill out a notification form to NSD and have it approved before conducting the research project. In the notification form I reported the type of data and which personal data that was collected, and described the research project in

general, and how data was going to be processed etc. My project is approved by NSD, validating ethical and secure collection, processing and use of data according to NSD's privacy protection standards.

6.1.2 Information to the Respondents and Consent

After I received an initial e-mail where the respondents had said yes to receiving more information about the project, I sent them an information note. In the information note I tried to make sure that the respondents had a good understanding of everything related to the project, and not in any way would feel, during or after, that they had been mislead. In the information note there was information about the topics and problem statement, the purpose of the project, how information will be processed and who will have access, how the results were going to be used and all consequences of participating. Due to the Covid-19 situation the teachers consented to participate by accepting to join an interview by e-mail (as opposed to a written signature), and in the information note it was stated that if they would participate, they participate freely, that they could resign from the project, even after the interviews, without further questions and that they had the right to access any data I had about them.

There is always a question whether an informed consent can be given in explorative qualitative research like this, because the researcher does not have all the knowledge about how the interviews will happen and develop (Kvale & Brinkmann, 2015, p.105-106). However, my overall evaluation of the project (approved by NSD) is that I have not collected sensitive data or information, and I have evaluated the questions not to be harmful to the respondents.

6.1.3 Processing and Storing Data

The storing and processing of data collected in the project is done in line with requirements and guidelines from NTNU (NTNU, n.d.-b), NSD (NSD, n.d.-b) and GDPR (GDPR, n.d.).

The interviews were done using *Zoom*, which is a software for video chat meetings (Zoom, n.d.). The software download and installation is provided by *Uninett AS* (Uninett, n.d.-a), who vouches for the security of the data (Uninett, n.d.-b). Zoom is also recommended for video meetings, academic conferences, etc. by NTNU (NTNU, n.d.-c). However, this was not without issues, because some counties in Norway have denied public sector employees to use Zoom due to claims about security issues through varying media attention (Carlsen, 2020; Brombach, 2020a, 2020b). Nevertheless, I have trusted the Zoom installation provided by Uninett AS, which runs on a nordic platform and server, and has stricter security requirements than e.g. the version in the US. Further details can be read on Uninett's website. In some special cases due to the formal issues with Zoom, the interviews were done over *Microsoft Teams* (Microsoft, n.d.) and *Whereby* (Whereby, n.d.), after mutual consideration and agreement about platform and security measures with the respondents.

The interviews were recorded with an external recording device, which was not

connected to the computer and only able to transfer data by cable. After the interviews, the recordings were transferred to an encrypted memory stick, and physically locked inside. Names and contact information pointing at the recording was replaced with a code saved on a name list separated from the other data.

The recordings were later transcribed. In the transcriptions I have removed all information that can be used to recognize the respondents. I have removed names, the school they work at, if they have unusual combinations of subjects they teach, if they use unusual programming languages or programming tools and other information that separately or combined with other information could contribute to recognize the respondent.

6.1.4 Reporting the Data

As the data has been anonymized, the report takes confidentiality into account. I have translated the quotes to English. It is important for me that the presented results, the discussion or any other part of the report, but especially the quotes, does not make the respondents or teachers as a group look bad. Of course, teachers can do mistakes, lack knowledge or have bad practice, and I would not exclude it from the report, but I have tried to be precise in my reporting, not generalizing and being careful in securing the respondents confidentiality.

6.2 Quality of the Research Project

I will here discuss the quality of the projects, mostly in terms of *validity* and *reliability*. Validity relates to whether a method is suitable for the research project, or in other words, whether a method really investigates what it is supposed to investigate; if the interviews, recording and transcriptions, together with the supporting literature and background, really reflect the phenomena we wish to investigate and research (Kvale & Brinkmann, 2015, p.276). Reliability is about the consistency and trustworthiness of the research, and is often discussed in terms of whether the results can be reproduced at different times by different researchers (Kvale & Brinkmann, 2015, p.276). For this research project, it means whether the results would be different if someone else conducted the interviews given all the information provided in this project and its appendix. Questions considering reliability and validity reaches beyond technical issues, but raises epistemological questions about whether knowledge which is produced from interviews can indeed be objective (Kvale & Brinkmann, 2015, p.272). I will here discuss some aspects of the data collection affecting the quality of the research project.

Considering the type of interview, an alternative type could be a focus group, where the interviewer works a a moderator for a group of people instead of one person (Kvale & Brinkmann, 2015, p.179). I considered doing group interviews, or the combination of personal and group interviews for the project. My argument for settling on personal interviews was that the perception of challenges may feel personal, and that perhaps the respondents would be more open to describe their challenges in

a personal, anonymous interview with only one person (Kvale & Brinkmann, 2015, p.179-180). On the other hand, I think especially for the questions regarding practice, there could be some rich data from a group interview where the teachers could build on each other and fill each other in with spontaneous and expressive views (Kvale & Brinkmann, 2015, p.179-180). Therefore, for some time, I considered doing both personal interviews for the challenges and group interviews regarding the practice. However, I wanted to relate the specific practice of specific teachers to their challenges. Also, this was a matter of time and resources, considering the scope of the project. I feel confident that deciding on personal interviews the way they were carried out was a good choice, but there are certainly other interesting ways of doing it.

In the data collection chapter I discuss how recording only sound is an abstraction of the real conversation with the teachers. A video recording would include more information, such as body language and facial gestures. Not doing video recording was a practical matter, because primarily it contains more sensitive data and is more difficult to get approval of formally without very good reasons to include visual data, and secondary because analysis would take more time and be beyond the scope of the project.

Interviews vary regarding how open they are about the purpose - the interviewer can explain the purpose and ask direct questions from the beginning, or ask more indirect questions and reveal the purpose after the interview (Kvale & Brinkmann, 2015, p.162). In my information to the respondents before the interview, in the briefing and the questions themselves, I was open about the purpose, the topics and what I was going to ask about. This was a choice based on ethical considerations, and I evaluated the possibility of getting more spontaneous answers not worth the ethical dilemma of having the teachers participate based on well informed consent. The topic is also quite narrow, and I saw it suitable to prepare the teachers with background information. The only aspect being indirect, was the way I was building up to some direct questions with more indirect warm up questions. I have explained and justified this in chapter 5.4.1 which concerns the interview guide.

Another important attribute to the validity of the results which I wish to discuss, is the fact that the teachers teach different subjects, which needs to be taken into consideration. For instance, the subject *Programming and Modeling X* is more concerned with numerical algorithms and mathematical aspects (Utdanningsdirektoratet, 2017), than *Information Technology* 1 and 2, which are more concerned with media production and web pages (Utdanningsdirektoratet, n.d.-b). It's is reasonable to assume that this affects the attitude towards code reuse, and the respondents indeed also mentioned this in the interviews. I have made an effort to take this into consideration, and interpreted the results with the different nature of the subjects in mind. I think a challenges with code reuse is a challenge with code reuse, even when the goal is a numerical algorithm or a web page, but it needs to be taken into consideration of course. The attitudes towards code reuse might be different. Being open to different subjects was also a practical issue, as I interviewed every teacher who responded to my request, and would have less respondents with stricter requirements.

Though hurting the reliability of the project, it is an advantage that I myself conducted all the interviews, and did all the transcriptions, because I was then able to evaluate the social context, remember moods and body language, and in general was able to reproduce the interviews as correctly as possible in written form with the transcriptions. There is nevertheless a trade off here between being close to the data material, but also not having anyone control my statements and interpretations properly.

Confidentiality is often considered in terms of anonymity and privacy for the respondents, but it is important to mention that it can also work as an "alibi" for the researcher by giving the opportunity to interpret the statements of the respondents without being contradicted (Kvale & Brinkmann, 2015, p.106-107). Due to the scope of the project, I have not went through the results with the respondents prior to writing the report. Doing this would improve the research in terms of the validity of the results.

In general, I have done my best to be transparent in how I have done things, and I believe that the research project could be reproduced. However, me as the researcher is unique for the spontaneous questions in the interview. I have tried to lean on state of the art literature about research methodology and qualitative interviews. As this is a student project, I do not have credibility as a researcher. However, I have thorough knowledge about programming, pedagogic, didactics and education, and I have also been through research method courses and had training in interview studies.

7 Analysis

In this chapter I have summarized the process of analysing the data collected.

The analysis begins already when transcribing, and the interview conversations are structured in a way more appropriate for analysis. When the researcher transcribes their own interviews, they will likely remember aspects of the interview situation important for the knowledge production, and the structuring of the information itself is a beginning of the analysis (Kvale & Brinkmann, 2015, p.207). When transcribing, I started to have several ideas regarding the results and later discussion, and kept a log during the work.

The analysis was done with a software called Nvivo, which is used to organize, store and analyze data (Alfasoft, n.d.).

Analysis in grounded theory is done in three steps of coding, and the aim is to generate theory to explain the data collected. The three steps are called *Open coding* to find conceptual categories in the data, *axial coding* to find relationships and interconnect the categories, and *selective coding* to conceptualize and account of the relationships through finding core categories or sometimes *the* core category to explain the theory (Robson & McCartan, 2016, p.481,483). I have explained what a code is in the following sub chapter.

In this project, due to the scope and the nature of the research questions, I have not aimed at identifying one grand category or theory to explain all the phenomena. However, I have identified categories, connections between them, and some conditional and contextual core categories realising them. As mentioned earlier, I am also not drawing out *the theory of formative assessment and code reuse*, but identify some challenges that some teachers experience when teaching programming in terms of code reuse, and suggestions for formative practice to overcome these challenges.

In the following sub sections I will go through the analysis process.

7.1 Open Coding

In open coding, the data is split into different parts or pieces, where each piece of data is considered a code and is an example of something particular amongst the data (Robson & McCartan, 2016, p.481). I went through the interview transcripts, and marked sentences, paragraphs or parts of conversation back and forth, and gave each part a descriptive label, i.e. a code. Open coding is about interpreting the data and find theoretical possibilities, and a piece of data can have several codes, it can be changed and it can be left out later (Robson & McCartan, 2016, p.482). I intended to code as much as possible of the data and try to sort it into different codes. Some of the codes came directly from quotes in the transcripts, while other were more descriptive with examples from the transcripts. An example of the latter would be a code "teacher encourages code reuse", when a teacher perhaps had said something like "I try to tell my pupils that code reuse is common practice and that they should reuse as much as possible". An example of a code directly from the example code

could be that "teachers think pupils should reuse code as much as possible". After the process of open coding, i ended up with 64 codes possibly describing and explaining the phenomena. At this point, many of the codes were overlapping, not relevant etc.

Even though the codes and categories arise from the data, the 'conceptual baggage' of the researcher, i.e. the theoretical background on formative assessment and code reuse inevitably influenced what I was seeing in the data. (Robson & McCartan, 2016, p.481-482)

7.2 Axial Coding

In axial coding, the researcher intends to link together the categories or codes from the open coding, putting together the data which was formerly split apart (Robson & McCartan, 2016, p.482). How this is done varies a bit for different research paradigms. I follow what Robson and McCartan refers to as an *interactionist paradigm*, where the coding leads to an understanding of the phenomena in terms of the context, conditions, actions, interactions and consequences related to the data (Robson & McCartan, 2016, p.482).

First, I tried to sort out which codes were most, or clearly, related to challenges and which were related to best practice. Some codes were labeled both as challenge and practice as they, as one can expect, were connected. At this point, I had nine codes under challenges, 18 under practices, and the rest not yet categorized further.

Then, I tried to code what could be considered contexts, conditions and consequences for the different phenomena, i.e. the challenges and the practice. The aim was here to try to identify how the codes were connected, and which contexts and conditions would realise or allow the challenges and practices. As for consequences, this contributed to understand the interconnection between challenges and practice; there were contexts and conditions where challenges happen, and consequences of the challenges leading to data considering practice. At this point, I still had nine challenges and 18 practices, and eight codes I considered contexts or conditions.

Finally, I analyzed each challenge and practice carefully to split together similar ones or divide significantly different ones. I also drew clearer connections from the contexts and conditions, and the challenges and practices. Finally I made several code which I considered interesting yet not necessarily directly answering the research questions, and some codes which were not interesting or necessary anymore. At the end of the axial coding I had five categories of contexts and conditions, four challenges, five practices and a more general category of formative assessment practice in programming. See results for details in the next chapter.

7.3 Selective Coding

In selective coding, the researcher selects a core category, which should be the basis for all the other codes or categories and be the foundations for the theory developed (Robson & McCartan, 2016, p.483). However, many studies are most concerned with exploring and describing the phenomena studied, such as this project, and the

axial coding has already completed the analysis (Robson & McCartan, 2016, p.482). With regards to the problem statement, the research questions and the nature of this research project, I have considered it sufficient to identify the different codes, their context, conditions and consequences, and how they are interconnected. However, i have sorted them hierarchically, with conditions and conditions realising challenges, which again prompt consequences and need for practice to overcome the challenges. In grounded theory, where there remains more than one central integrating aspect, it can be argued that they can always be integrated to a single focus at a higher degree of abstraction (Robson & McCartan, 2016, p.483). Nevertheless, given the scope of the project, the research questions and descriptive and explorative nature of the study, I didn't find it appropriate or necessary to try and formulate a "grand theory" for the challenges of formative assessment. It can also be difficult in practice to decide when categories are saturated and the theory is fully developed (Robson & McCartan, 2016, p.482), and to make more certain claims I would have to revisit the field, i.e. the interviews further. The hierarchical arrangement of the codes found in axial coding was where I completed the analysis. The results are presented in the following chapters.

8 Results

In this chapter I have presented the results from the analysis. I will here use the term *achievement level* of describing the level of a learner's skill, knowledge, competence etc., because it is close to the Norwegian term *grad av måloppnåelse*. I will also proceed using the term *learner*, because it can generalize to pupils, students or someone learning not related to formal schooling. However, in the interview quote translations I have used the term *pupil*, because it is closer to the Norwegian *elev*. I have used the term coding *manually* and coding *from scratch* to describe coding practice where the writing of code is done by a person themself, instead of either the code itself or the code structure being copied from somewhere. However, from a computer science point of view, "coding manually" can be ambiguous. I have also described that some teachers have a *positive* attitude towards code reuse, while others have a *restrictive*; this is not to be misunderstood as a biased good or bad attitude, just that they are positive to the concept in classes or more restrictive.

On the next page is a figure illustrating the results. The top layer starting with 8.1 represents the contexts and conditions allowing code reuse to be a topic of interest and challenge in terms of teaching and formative assessment. The next layer 8.2 are the challenges as described by the teachers. Then there are the practices to overcome or mitigate the challenges marked 8.4. I have also included general formative assessment in programming as 8.2. I cannot claim that they directly mitigate the challenges presented, but are a part of the teachers' formative practice.

Figure 3: Results

8.1 Context and Conditions

I have tried to identify contexts and conditions as to when or why challenges and practice happen. For code reuse to be a challenge to formative assessment, it is relevant to understand why it it is a practice and why it is allowed to happen in the first place. Put bluntly, a solution to challenges to code reuse would probably not be to remove code reuse from programming practice in classrooms.

8.1.1 Code Reuse is Common Practice and Encouraged by Most Teachers

One of the main reasons reported by the teachers as to why code reuse is a common practice and encouraged, is that they have the impression that reuse is a common practice in real jobs involving programming. One of the teachers explains that *"I know several people who work in software development businesses which, who use standard templates and develop them. So why should the pupils then do the same code a thousand times through the school year here then?"* (T6). The teacher is saying here that in real programming jobs, they develop from standard templates, as an example. Many of the teachers share this point of view, that the way the learners do their work should be applicable to how developers work in real life. T1 says that *"If you are going to live of this and make money of it, then [code reuse] is a good way of doing it"* (T1).

With code reuse being understood as a common practice that the learners should do and learn, most of the teachers from the study encourage it. Teacher T6 explains that *"there are many examples. And a lot of such, we can take an example from [W3Schools], and then we build on it given what's asked in the [text]book. And that is because we don't have, don't need every time to re-invent the [wheel], so to say"* (T6). The teacher explains that they work with reusing examples they find online or in textbooks. T1 says that *"It is a little up to the pupils, but I say that, like, learn how to be lazy and indolent programmers, because then you become more efficient if you sort of don't do everything yourselves, but use what exists"* (T1). Many of the teachers also explain that they encourage code reuse.

Amongst the 12 teachers interviewed, only one had a generally more restrictive attitude towards the learners' code reuse. Summarized here, the teacher said that he wishes that the learners should learn from scratch and develop proper understanding, before doing "fancy" programming with a lot of reuse. It is worth mentioning that the teacher teaches one of the more mathematical programming courses "Programming and Modeling X", and it would have been different if he taught something more applied. However, there were other teachers teaching the same subject but were very positive to code reuse, so there are different practices amongst teachers as well.

8.1.2 Code Reuse Allowing More Advanced, Interesting, Motivating and Efficient Work

An important motivation for including code reuse in the learners' programming practice, and teaching them to reuse code, is that they might be able to work with exercises and tasks which are more motivating, varied, interesting and advanced, and they can do so in an efficient way.

When not having to write all the code from scratch, and being allowed to use some code, libraries and interfaces that are over the learners' skill level, but still has learning gains, the learners can work with more motivating and interesting examples and exercises. Teacher T7 explains that *"[the pupils] had projects where they program and create games, and then they use built-in libraries like [library name], for instance,*

and that... Where others have made games, and then they use it as a starting point to create something themselves, and then, then there will be reuse of parts of the code, and we rather look to what we can make our own" (T7). The teacher here is saying that by reusing certain libraries, the learners can develop games, because they get some foundation to start developing from. Developing games is more advanced, and can be more motivating and interesting, than developing something they have the achievement level and time to develop from scratch.

Besides from skill level, it takes time to write programs from scratch. Many of the teachers embraced code reuse as a practice because the learners were able to work more effectively and efficient. Many of the teachers mention that the resources do exist, so the learners should use them. Teacher T2 says that *"they are able to work more or less fast and efficiently, and the thing with using the resources they have available is perhaps even more important than in other subjects"* (T2). Not only can the learners work more efficiently for the sake of it, but they are able to do more, see more examples and increase learning. Teacher T6 claims that

> Code reuse can promote learning, in the sense that they can create more stuff in less time. That they have a template they use each time so that they don't need to spend fifteen minutes to write that template, so that they can shorten the time to do an exercise or make a product (T6).

The teacher is explaining here that if the learners already have a template, they can skip writing, creating or doing a certain task many times, and instead skip to the new parts of learning. T1 adds that *"The advantage is that they learn to steal, and learn to be more efficient. Yes. And not do what others have done, that is, not re-invent the wheel. If it already exists for legal reuse"* (T1). There is thus an attitude towards code reuse as efficient so the learners can do more work in less time, learn more, and not repeat work that is already done.

8.1.3 Making it Feasible to solve programming problems without knowing how they are solved

Closely related to the result above, one aspect allowing code reuse to become challenging, is that solving programming exercises, tasks or problems is feasible by copying without really understanding the solutions or knowing how they are solved. This can be challenging in itself if learners choose to copy solutions where they are supposed to learn or practice something, and also if they choose to work on problems or solve exercises which are too advanced for them, because they do not understand their own skill level. Teacher T7 elaborates

> Quickly they want to do very big things, right, the creativity easily runs a bit wild (...). And they are left with a product where they don't really understand why the different parts are as they are (...). That they don't really understand what their program does. They can run it, but that's it (...). And then, clearly you have been able to find relevant information,

> and that's a good skill to have, and you have been able to fit the information together into something, but if there was to be a problem, you may have had big issues with fixing it (T7)

Teacher T7 is explaining here that the learners sometimes work with too big and too advanced projects because code reuse allows it, but are not able to understand their own work.

8.1.4 Teaching to reuse code

Besides from encouraging code reuse as presented in 8.1.1, some teacher have an explicit focus on teaching code reuse, and that code reuse as a concept itself should be practiced. However, other teachers had less focus on this because it is such a small part of the curriculum, while others again integrate it in practice but not explicitly to the learners.

For some of the respondents, teaching to reuse code and how to do it appropriately was an explicit focus, aim or goal. When asked not only if code reuse is a practice among the learners, but if the they had a focus on *teaching* code reuse, teacher T10 replied

> In exams in IT2 there is actually a specific goal that touches reuse of code. And there, I applaud it, if they are able to show that they can adjust it, and understand it, and that they state references. That are kind of the requirements for reuse of code which I put in the assessment, and in, I think that code reuse, you understand how it works, and you state references orderly, then it's a satisfactory solution as if their peer would code it manually. Because then they show competence which is important for a computer scientist, namely working with unknown problems, and everyone (...) are 'shopping' solutions online. But then I do an evaluation, this I can use, this is worthless. An that's sort of, that's where the competence is. (T10)

T10 explains here that he considers a solution with code reuse in many situations as equally satisfactory as a solution coded manually. He mentions that it is a goal in the curriculum, and explains that learners should learn to evaluate the code to be reused, reference properly and understand how it is supposed to be used. He also elborates that he won't only teach the learners to reuse code found online, but also to make reusable code, so that learning the concept of reuse includes creating reusable code assets, and reusing reusable code assets. He says:

> It is our approach that they should produce their own code which can be reused later, meaning that they create libraries and classes. As such. But we also have the focus with being able to modify others' code, and that's also reuse to some degree, you know, analysing and modifying others' code would be the wording. (T10)

Teacher T4 provides another example where they had done projects where the learners had roles as orderers and producers of code, where the orderers would order pieces of programs to create a bigger system and the producers would produce the reusable code artifacts.

Most of the teachers, whether having a very positive attitude towards code reuse or not, had clear opinions of what was considered acceptable code reuse, or when the code reuse was done right, which is an important aspect of teaching the concept. Many of the teachers explain that reusing "parts" of existing code is accepted, and that "parts" of the learners' solutions could consist of reused or copied code. Teacher T5 explains:

> As recent as last weekend I assessed a [description] exercise, and then I saw something that obviously this pupil couldn't have done by themself. I see it slightly in the code, and then I googled and found a YouTube video with the exact same screen image, exact same wording on things. It was translated to Norwegian command prompts, but that was the only modification that was done. And then I consider it cheating, I mean, when you just take a whole project and pass it as your own with small modifications, then you show that you know nothing. But what I openly say is that we work like that for real, that we have internet as a source, we have to steal things out there not to need a lot of time, get to know things, or learn everything new, but then it's easier to take snippets of code, that is, that you take smaller parts. Never take large parts, because then you have trouble achieving the total understanding of how it is put together. But if you for instance need to learn how to read arrow keys in a game, then you retrieve code for it, and try to understand how it works, and then you try to include it in your own project. So taking, as mentioned, smaller parts, from wherever (...). But it's a little up to them, so I'm pro that they can retrieve a little, a little larger parts, but never whole projects

This teacher explains that one one hand, he has a clear message that you cannot copy whole projects, but smaller parts. What types, and to which extent the code reuse is accepted, varied between the teachers. Some had a very positive attitude where anything could be reused as long as the learners stated references, while others' would be restrictive for online code reuse but open for using built-in libraries for programming interfaces, reusing code from text books or from peers. There is also a difference between reusing the parts which are not directly related to the learning goals, compared with copying the solution to an exercise directly. To summarize, there is an aspect of the *amount* of acceptable code reuse (snippets, components, whole projects), and the *type* of acceptable code reuse (copy-and-paste, use of built-in modules and libraries, online/not online). From the interviews, none of the teachers claimed that there were specific rules or frameworks for what is acceptable and not, and that it was more of a rule-of-thumb or their evaluation of the situation.

Though varying on what is accepted, there seem to be some types of code reuse in the context of a classroom. These include reusing code given by the teacher or in the textbook, reusing their own or their peers' code, reusing built-in modules and libraries for a programming language, and finally online code reuse.

Several teachers mentioned that code reuse is not a significant part of the curriculum when asked if it is a topic they have considered in depth in terms of teaching programming.

8.1.5 Learning through code reuse

Many of the teachers interviewed, eight out of 12, claim in some way or other that code reuse, and even straight forward copying can have some learning gains, especially for the weakest achieving learners. Anything from looking at examples to get inspiration, to directly copying to just see something work can have good learning implications. T4 talks about both finding and copying code from online, and also using programming interfaces where instead of coding with text, the learners can retrieve pre-made "blocks" of code. He elaborates this:

> Especially this relationship with the popularity of block programming and these programs that help you understand functionality and logic and the building of structure, before you are punished for writing errors, is very useful, and a very effective arena for, for *mestring*. Because you quickly get to produce, and then you can just convert and see that, okay, but this is purple, blue, yellow blocks, okay, cool. And it does the work it should. You are going to practice in a programming language later, and then it will be given this way, and it's like we have a native language. The logical native language, and it lies in the structural understanding and the relationship between objects and functions, arguments and variables, and declarations. Finding those analogies is a practice in itself, and it's a challenge with understanding where the learner comes from. Which terms, structure, and really which logic understanding the pupils has in their way into this world (T4)

He explains that there are some structures to be learned in order to realise programming, and that the logical structures can perhaps more easily be learned through some sort of reuse.

When getting beyond the level of straight copying, the learners can also do small code modifications. Teacher T3 tells that:

> There's no doubt that taking code and changing it is a step into writing the code yourself. So I think that in beginner education in programming, you need to have code that you... That you retrieve yourself and use. And then it's easiest in the beginning that I give them something to modify. Where they, for instance, start with summing every number in a list, that

> is, here I have made code for it, and their task is to make their own code that finds the product of all the numbers in that list (T3).

He here explains that he can give the learners an example, and that they can do tiny modifications, such as change something from summing numbers to finding a product, to get a better understanding of for instance iterations through a list.

Even if the learners *are* able to write the code, code reuse can be a way of discovering new solutions to known problems. T10 says that *"Code reuse is also important, because it can also make you discover smart techniques which perhaps not the teachers or you have seen, or different ways of solving the same problem, which also is very important in programming"* (T10)

As we can see, there are many learning gains built into reusing code and copying others' solutions.

8.2 Formative Assessment and Practice in Programming Classes

In this part I will briefly sum up how the teachers in the interviews said they work with formative assessment in general when teaching programming.

One of the most common, yet also important, answers was that when speaking of formative assessment in programming classes, the teachers think of it as something they do "all the time", and in-process. They describe formative assessment as a continuous assessment work in every part of class. Teacher T5 explains:

> "They will have a summative grade on the product they deliver anyway, or a summative assessment, a comment, but because they use a lot of the classes working, I circulate almost continuously and see what the individual pupil does, and give feedback, and push them the right direction. Typically with little hints first, and then, if they then proceed, that's great. If not, I sometimes need to sit down a little more, and, together with them, all the time intend that they suggest the solution to move further (...). That supervisement technique, that's sort of assessment for learning. Continuously." (T5).

In this example, T5 explains that he uses every moment to speak with individual learners to both evaluate how they are doing and pushing them the right direction, i.e. formative assessment. Many of the teachers had this attitude towards formative assessment in programming class.

The second most significant answer to general formative assessment, was the use of conversations and discussions. This includes discussions, either full classroom discussions, one-to-one with the learners, with groups of learners or that learners are discussing with their peers. Teacher T2 discusses classroom conversations related to the Covid-19 isolation situation:

> We actually see it really well in this situation we're in now, and, you should have thought that programming and coding was a subject that was easy

> to bring home and sit each by themselves doing, but I experience maybe that this is the subject where the pupils find it hardest not to talk with each other. Because they are so used to sit and discuss solutions, discuss progress, or how they are going to solve a problem. That they work even more together perhaps. And that they find it harder to achieve when they don't... Yes. They can of course sit and have the same, they can share a screen and all that, but they kind of lose the possibility to go to the neighbor group and ask... Yes. So I'm a bit surprised by that.

Teacher T2 is here surprised about how important conversations are for the learners in programming. The teachers also tell that all types of oral discussions are important, and that it's an important factor when trying to understand the achievement level of the learners.

A third common topic with regard to general formative assessment practice, was the use of open exercises. Open exercises are exercises in which there are several ways of solving the exercises, and that the exercise has several correct solutions (Karlsen, 2014, p.36). They are often mentioned together with *rich exercises*, which have the same attributes as the former, but in addition are designed in a way that the threshold for solving it partially is low, and at the same time solving every aspect of the exercise is cognitively challenging (Karlsen, 2014, p.37). The teahcers explain that open exercises helps them differentiate levels and working with formative assessment and feedback depending on where the learners are in their learning process, while working with the same exercise or project. Teacher T5 says:

> It's about spending time to make exercises in such a way that they can land it on the level they are. In other words, making a basic version, and then they can make something more advanced on it, and then they can make something straight up difficult on it (...). Make exercises which are, which have room for expansion, but starts low, so you have the opportunity that everyone sets the bar where they wish to be. And then it will also be that they express where they wish to be in terms of grades. And someone just wants to pass" (T5).

T5 explains here that the learners can work with the same exercise, but at their own level, and engineering exercises like this is important for the formative assessment work.

Related to the statement above, there is also a formative practice of letting the learners set their own goals. Teacher T6 tells that in the beginning of a course he will ask the learners:

> What do you think? [the teacher asks the learner] Here are the learning goals for the subject, and here is what we are going to work with [this semester]. What are your thoughts here, what do you sort of want to, what do you think is most achievable for you, what are your goals? And then sum up at the next talk [of this kind] at [the end of the semester]. How did you do, and how has it been (T6).

T6 describes here a formative practice allowing the learners to affect their goals, and decide on what to do depending on their current achievement level and goals.

As a last distinct way of working with formative assessment, some of the teachers mention portofolio assessment. Portofolio assessment is when all the learners work are put in a "portofolio", so that not one exercise or project but the whole of a learners work can be included when doing assessment (Engh, 2017, p.149-159). The portofolio can include the learner's goals, logs, reports of conversations with the teacher, self-assessment, results from peer assessment, exercises and tests (Engh, 2017, p.149). Teacher T10 elaborates:

> "I've had great interest for more explorative ways of working and more formative assessment methods, so I started quite early by thinking in terms of portofolio assessment, and that they can also work iteratively with tasks, meaning that I can let the pupils produce a solution, then have, then we can have a dialogue around it, then they improve it, then we have a new dialogue, they they improve that for instance and... And now I think more in those terms, and I think it works well. And by combining this with open exercises so that the pupils can work with the same problem, but at different levels, then I think we have a good foundation to work formative. So my aim is to cut out tests completely." (T10)

He explains that portofolio assessment is good because the learners can iteratively deliver their current work, and do it better in a next step.

Regarding these different types of assessment as separated from another is not possible, and everything will be connected to the other in many ways.

8.3 Challenges

I will here present the challenges with formative assessment with regard to code reuse that came up during the study.

8.3.1 Learners Reuse Code Which They Do Not Understand

When the learners reuse code they find online, in their textbooks, given by teachers or borrowed form their peers, it becomes a challenge for the formative assessment when the learners do not understand the code they are reusing. This is a formative challenge because the learners are unaware of the goals, or skipping the goals, and cannot do proper self-assessment or work with assessment with the teacher. This is one of the more significant challenges, mentioned by nine out of the 12 teachers interviewed.

This happens when learners are able to find code, or a code solution to a problem, which they implement without understanding it. Teacher T10 tells that *"I can see it in many of the exercises that the pupils often find a lot on Stackoverflow, and in a way gets to solve the problems there. The issue is that very many of them don't have a concept or idea of how they use the code, they don't have ownership to the code"*

(T10). The teacher explains that this happens when the learners just find a solution and use it without trying to understand it. It can be that the learners do not do an effort to understand the code, but sometimes the retrieved code is just too advanced for them. T9 says *"[on Stackoverflow], it often becomes a little too smart, thath what is supposed to proceed, and it's a little too difficult for them to understand what it is we're doing"* (T9).

Besides from copying solutions directly, the learner might also make small adjustments to retrieved code, but still not understand the main concepts. T10 explains

> Many of them can take a pretty advanced example and just wrench it if they understand on the variable names and such, that if I adjust here and change there, I make it work, but, they have no idea how it works. And I constantly receive examples on deliveries where they in fact have solved the problem (...). But, then they struggle a lot when they are going to explain the code, and that is a good example of how it hits back, this code reuse (T10)

Here, the learners are able to make adjustment and deliver apparently good solutions, but have indeed not understood what they have done. One of the main problems doing this is that exercises are apparently solved, solutions are apparently good and programs work and do what they are supposed to. T11 tells that *"as long as things work, and that they get done what they are supposed to, that it satisfies what they are supposed to do, then it's fine. But, of course, it is not the same as if they have understood what they have done"* (T11).

Some consequences of this challenge is that assessment for the teacher of a seemingly good product can turn out difficult, but also the learner's self-assessment hurts from this, and they may not be able to understand what is wrong with their programs when running into errors. T2 says *"Because then you are not able to fix mistakes, and... And we have had some examples on as well, that they've copied code that they, which in fact lies much more advanced than what they really are able to, and then they are not able to understand what doesn't work"* (T2). So this challenges hurts both teacher to learner assessment, and learners' self-assessment, because the code becomes too advanced for the learner.

8.3.2 Challenging to understand the learners' general level of achievement in relation to code reuse

Besides teachers having challenges assessing learners based on certain exercises, or evaluating certain exercises, a challenge arises related to formative assessment with understanding the general current skill level of the learners. This, again, touches both the teacher having challenges interpreting what the achievement level of a learner is, and the learners themselves understanding their achievement level. One of the teachers tells that learners can be good at reusing code but bad at coding from scratch manually, while others are good at the opposite the opposite, or anywhere in between. He says

> On one side, you have someone who can copy a lot of code and push it together and make everything work, and that it kind of, it is a really good final product. He has copied a lot of code, not written anything himself, but he has understood everything and knows how to relate things to each other. But if I ask him to write all that manually, he will be absolutely stuck. Then, on the other side, you have someone who writes everything manually, but has some trouble with kind of putting things together and understanding what is related. But he is very skilled in writing many codes (...). It becomes a balancing act and a matter of assessment and assessment for each pupil, really (T6)

The teacher explains here that both types of programmers can be good, and that the teachers needs to do proper assessment to understand the learner. T6 also adds that for each side of the spectrum, the learners might as well know equally much, or that he evaluates them as equally good programmers. So evaluating the general achievement level of a learner can prove challenging, especially in terms of code reuse, because it opens up doors to program in very different ways.

Moving on, many of the teachers from the interviews, six out of twelve, claim that code reuse is a skill at a high achievement level for the learners, and that code reuse in fact indicates a high level. Teacher T1 says that *"you need to be at a certain level, you know, to be good at reusing others' code. And you need to understand the code (...), understand what you are doing, and when you get there that you sort of can, are good at retrieving code from others, then you're already at a pretty high level"* (T1). The teacher claims that a condition for code reuse is understanding. The teacher continues by explaining that *"reuse of code is really an advanced technique (...). You need to have designed a solution, know what it is you are going to create and what you need (...), what types of functions you need, which methods you need, I mean, how you are going to build your solution"* (T1). It seems as if code reuse can indicate a high achievement level, regarded that the learners design solutions, modify code, retrieve the right code from the right places and so on. Some of the other teachers also explain that good code reuse means the learners have a good overview of programming and their problem, knowing what they are looking for, modifying code independently, using small parts in bigger programs and are efficient in their reuse. When teacher T4 claims that code reuse can be useful for lower achieving learners, he says it in a context where the teachers help the learners a lot, instead of the type of independent code reuse described above. He says

> We set up the structure, we hand them the functions, we split up what the task is; you are going to expand this code for instance. And then, to expand existing code forces you to get to know the code that's there, and then to see what lies, what must, what needs to be changed, that it can be easier than to see that, okay, here is a blank page and a challenge (...). So the lower achievement level the pupils fundamentally have, the more [scaffolds] and frames we need to set (T4)

Thus, there is not necessarily a correlation between code reuse and achievement level, but different types of code reuse can indicate high and low achievement of learners. But not all of the teachers' statements coincide here as some describe code reuse as a high level skill, while others have the impression that equally high achieving learners can be good or bad at reusing code.

8.3.3 Code Reuse at the Expense of General and Specific Programming Training

Whether a teacher's attitude to code reuse is positive or not, the teachers in the study claim that code reuse can be a practice that is done at the expense of more general, "manual" programming training or practice. In eight out of 12 interviews, the respondents discussed this. Teacher T1 elaborates thoroughly about this topic:

> You can say that if there turns out to be very much of [the code reuse], then it's not so favourable, but it's my task to... My job is to give them exercises that makes them have to program some amount on their own. I mean, first and foremost they are supposed to learn programming, that's what the subject is about. To learn all, learn programming techniques, basic programming techniques (...). But, of course, if it turns out to be too much [code reuse], then it affects their training in programming, and training in solving the problems themselves. If you find a sorting algorithm online, you don't need to think too much about how it happens (...). If you find a sorting algorithm (...) where you can send in [a list of variables, e.g. numbers] into the sorting algorithm and receive it [sorted] back, then of course, then you learn exactly sending in and sending out, or fetch out, but you don't learn kind of what happens inside there (T1).

The teacher is explaining here that if learners are supposed to learn some sort of programming, they need to work specifically with that aspect. Good awareness about what they are working with is important here. From what teacher T1 explains, awareness about exactly what the learners should learn, or what the goal of an exercise or task is, is important. Many of the teachers in the study share this view.

Teacher T9 emphasizes that code reuse might not only affect the general programming skills, but especially if the learners reuse when they are supposed to learn a specific skill, this becomes a challenge. He discusses the implications of a goal being programming with pre-made functions or the goal being programming the functions yourself. He says:

> So, instead of kind of finding a smart solution to how you can print the first ten prime numbers by using generators and filter and this and that, such things, I want them to sit down and think about what is really a prime number, how can I kind of, what, which things do I need to do with the small framework I've got here (...), how can I achieve it with that - rather than how I do it sexy in Python" (T9)

T9 and the other teachers claim that if code reuse happens on the expense of specific goals or programming skills, this becomes a challenge for assessment.

8.3.4 The Teacher Does not Understand the Reused Code of the Learner

With exercises, tasks and projects allowing extensive code reuse, which in many cases is perfectly normal, it can become a challenge for the teacher not being able to understand the reused code, or that assessing or helping the learners becomes too time consuming because of this. Teacher T6 says that *"Yes, the challenge (...) if when code reuse or copying happens, then it can quickly become that if you don't pay attention as a teacher, that you're kind of not being involved in the pupil's work, then they may swiftly copy or paste codes that you don't understand yourself"* (T6). Another teacher explains the importance of being involved in the learners' work at all times, and says that *"because if a pupil works isolated with something for three weeks, and you then receive it in your lap, then it's... Familiarizing with the pupil's work can be very extensive"* (T7). So two challenges for formative assessment here are that the teacher can't help the learner because the code is too advanced for the teacher as well, or that understanding the code is too time consuming and the teacher needs to break down and build up again everything with the learner. However, they are painting out situations which they won't ususally find themselves in, like not having paid attention to a learner in three weeks.

8.4 Practice

Teachers' formative assessment practice to overcome and mitigate challenges presented above.

8.4.1 Tasks and Exercises

Many of the challenges arising from learners being able to copy-and-paste solutions to exercises and problems can be avoided by engineering better exercises. Teacher T1 explains how extensive code reuse can be challenging for assessment, but that *"my job is to give them exercises that makes them have to actually program to some extent on their own, that is, first and foremost they are supposed to learn programming, that's what the subject is about."* (T1). Many of the other teachers also discuss that they need to engineer good exercises, and can't blame the learners for copying and reusing where possible. T10 says that he needs to take the criticism because the exercises are in these cases bad. When asked how to avoid these issues he shows me an exercise where it is just an image of a *One Armed Bandit* game, and almost with no further guidelines, the learners will try to implement this. T10 elaborates:

> You should make open exercises, perhaps where the learners make up the problem statements themselves, or where there are several ways of solving the exercise. And I can see, like in the one armed bandit exercise - I have yet to see two similar solutions on those kinds of exercises, both regarding

> the level and how they implement the code. So that is not really a big problem, so I think the responsibility lies upon the teacher, that if you have an exercise where the answer is '4', you kind of know what you get from the pupil, but if you make up a more open and good exercise, I think both the pupils learn more, and the teacher also gets a better foundation to work with the subject onward, with regard to the achievement level of the pupil. (T10)

Simple, straight forward copy-paste problems will result in copy-paste answers, while exercises encouraging designing a solution, planning, breaking down the problem yourself etc., result in more versatile deliveries. The teachers in the study discuss how giving exercises which are easy to find solutions for online, which are almost the same as earlier exercises will result in the learners reusing older solutions, without really having done anything wrong, but with the disadvantage of not learning. When discussing that these very open exercises can be very difficult for the lower achievers, T11 said that *"at least as a minimum, that they have adjusted the content and variable names so that it's appropriate, that I can see that it belongs [there]"* (T11), and jokingly says that he has received solutions to exercises with variable names like "new car", even when the exercise was about something completely different from cars. He also said that *"there has to come in something new int he next exercise, or there needs to be a variation that demands that something needs to be changed in the code that is being reused"* (T11). Exercises need to change, and need to require some modification of either retrieved code from online or reuse of earlier solutions.

8.4.2 Continuous Assessment

Many of the challenges are overcome by the teacher being active, involved, and doing continuous formative assessment of the learners, focusing on that assessment is conducted all the time through all parts of a class, not just the evaluation of exercises. Teacher T1 tells that

> Most of the time you know where the pupil stands. You walk around in the classroom and see what they are capable of, and we help them kind of all the time, and see what sort of code they write, and we have fairly... I think you have fairly good control over where the pupils stand (...). My advice would be to be with the pupil in the class, and watch what the pupil is doing in class (...). and discuss. Discuss the code with the pupil. Discuss the job they have done. (T1).

Teacher T1 emphasizes being a part of what the learners are doing and being active, discussing and assessing. The other teachers also mention that they try to do assessment work all the time by asking questions, looking at the programs, asking the learners to upload screenshots etc. Teacher T3 explains that

> Here comes the 'investigate all the time' thinking, that you need to speak with the pupils about, okay, how were you thinking when writing the

> block, why did you do this instead of that, and then it's smart to ask about details, the little weird, quirky things (...). You quite simply need to try to have an overview over the competence of the pupils, even though, of course, the pupil develops all the time, so you cannot just say that no, the pupil doesn't understand programming that well, so I don't think he can do it that well. I mean, you have to update the thinking all the time, about what they are able to and not (T3).

The general rule of thumb here is to think formatively all the time, and conduct formative practice in every situation. T5 says *"I mean, that's the full time job of the teacher, to... Even when I hear questions they ask in the classroom, I understand if they have understood the point. Or, you know, or if the questions are at such a low level that they haven't understood the point"* (T5). If code reuse for any reason makes the teacher "lose track" of the level of achievement of the learners, thinking continuous formative assessment might help.

8.4.3 Assessment of the code reuse in particular

Another practice to overcome issues not understanding the knowledge of the learners because of code reuse, is to work more specifically with code reuse and including it in the formative assessment work. All the teachers in the interviews mention in some way or other that the code reuse needs to be conscious, thought through and with awareness part of the programming practice. Teacher T7 explains:

> The most important for my part, I think, is playing with open cards. And, like, not have a focus on this thing that using others' work is, that it's cheating or... I mean, not set restrictions, but all the time encourage that, okay, but if you are using someone else's work - like, why? Can you support, what is it that makes you take exactly this choice among thousands of examples around the internet for instance, or that, that you always encourage them to think and reflect around their choices, if they get inspiration or parts of their code from others, that is (T7)

T7 explains that code reuse should be done, but honestly, consciously and with evaluation of the reuse. The teachers also claim that assessing the different parts of a code reuse process contributes to making code reuse less of a challenge for the assessment. T10 says *"we have exercises that, where, like, this about search and evaluation as well, that is, information evaluation, and assessing that is important as well"*. He says that assessing the evaluation of the information to be reused is important, along with other parts of the process like modification of the code and integration of the code in own programs. When asked about code reuse and assessment, T4 answered *"[The code reuse] definitely affect how I assess them. If they haven't done anything themselves, but retrieved prepared code, then I need to assess the degree of adjustment, the degree of modification of what they have done, which shows if they have understood it"* (T5).

8.4.4 Code Modification for Understanding

Asking learners to modify their code specifically was reported by a significant amount of the teachers as one of the most effective ways of really understanding if the learners understood their own code, and evaluating their general programming level of achievement. It is a very specific practice, but claimed to be so efficient that it's presented as an isolated result. Teacher T1 tells about formative assessment: *"we go through the code, they explain what they've done. And the next thing we do, is that you ask them to do some modifications so that we see that they are actually capable of understanding what they've done. And then you sort of can control that they have the competence"* (T1). He explains that when assessing learners, the modification of code reveals a lot of understanding. He says that *"yes, because then you can see very well which kind of understanding they have, and what they're actually able to"* (T1) and he explains that it's plausible that the learner has just copied, i.e. reused, mindlessly if they fail the modification test. T4, when speaking about code reuse being at a high level, he says *"So producing code is one element, but recognizing and understanding how structures work and possibly reveal what kind of final product a given code gives, that is a different degree of understanding. It seems intuitive to me and natural to see that, okay, if you can read the code at this level, then you can also explain to me what to need to modify so that it solves a problem int his way or that way"* (T4). It seems as if monitoring the learners' code modifications can be a particularly good way of gaining understanding of their achievement level in programming, and understanding their particular code reuse situation.

8.4.5 Multi Variant or Multi Dimensional Assessment

In general, not just assessing particular exercises or particular aspects of learning, but performing formative assessment of different dimensions of the programming and code reuse, is one of the most clear ways of truly understanding the learners. All of the teachers mentioned this. They include going through code with the learners, having conversations and discussions, do modifications, create documentation, present their work in presentation, screen recordings - in general try to find different dimensions of their work to assess to gain understanding. For instance, teacher T10 is very fond of learners creating video presentations with screen recordings to show new dimensions of their work. He says:

> And at the end, as a delivery, they need to record a screen recording, max three minutes, where you show and explain the program. please also explain what was difficult or you were not able to do as well. And then I identify immediately those who have code which is reused that they don't understand. It is revealed instantly, and we can have a conversation about it. While others, who show more confidence, either by creating their own code or reusing code that they don't understand, they reveal as much in the recording (...). Over some time I have a whole lot better foundations for assessment than what I would have from an exam or a test" (T10).

9 Discussion

In this chapter I will discuss some of the results presented over related to formative assessment, and teaching practice in general.

9.1 Teachers' attitudes towards code reuse

An interesting theme emerging from the interviews is the different types of attitudes towards code reuse that the teachers seem to have. When given the chance to reflect on code reuse in the classroom, it seems as if most of the teachers' descriptions of positive traits of code reuse coincide with the ones described in the theoretical framework; programming becomes more efficient and effective, there is less need to re-do time-consuming work and there is a possibility to create code and programs that are more advanced and complex, compared to what would be possible without the reuse.

As for attitudes towards code reuse, and perhaps level of awareness, a few of the teachers interviewed claimed that they actively taught the learners to reuse code. They did learning activities where the learners were supposed to consciously and with awareness retrieve relevant code, understand and evaluate it, modify it, and integrate it in their own programs.

The majority of the teachers from the interviews had a positive attitude towards code reuse and encouraged learners to use available resources as a rule of thumb, but did not consider code reuse per se as a concept to be taught and focused on. It seems like the majority of the teachers from this study think of code reuse as a normal phenomenon to happen when coding, but that it is such a small part of the curriculum that it is not something they work on explicitly. This is interesting because it fits the general attitudes towards code reuse as something that is integrated in programming, and that "everything is a remix" anyway, however neglected the purposeful work and use of code reuse.

The third distinct attitude towards code reuse from the interviews only came from one teacher, who was very restrictive towards code reuse when learning programming. Though only one out of 12 teachers, I think the reasoning is particularly interesting, important and relatable to other situations. Also, given the nature of this study, none of the results can really be considered significant in terms of true generalization. Now, the teacher who was more restrictive with respect to code reuse, was more focused on teaching the learners the basic concepts, basic ideas and mechanisms behind programming. As he was teaching a more mathematical programming subjects, he wanted his learners to truly understand how and why the numerical algorithms worked as they do, the mathematical ideas behind the solutions and the programming logic. He also discussed how he didn't want to teach the learners a programming language, but *programming*, and that he wanted the learners to use as little helper functions and modules, external resources and, to put short, code reuse as possible, in order to learn. What I interpret this kind of attitude or paradigm towards learning, as is

that learning should happen deductively, from clear lower steps to higher, or with building blocks from bottom to top. I think this is a recognisable position, especially in the natural sciences, where for some teachers learning should start at some logical point, and deductively emerge. The other teachers have perhaps a more applied, inductive position, where the learners can see and use what is available and possible, try out things without understanding, with learning understood more interconnected yet without structure. As I will discuss in 9.4 which concerns goal attainment, there are not clear frameworks or guidelines as to how code reuse should be used, and if learning should happen in a deductive or experimental manner, and thus it is hard to claim that some of the teachers are right or wrong about this subject. I will also discuss further the reuse in programming compared to other subjects in 9.9. It is also worth mentioning that though only one teacher in this study was being very restrictive towards code reuse, almost all the teachers claimed that reuse as a practice can happen on the expense of general and specific programming training.

9.2 Challenges and Practice Consider Both Teacher and Learners

The challenges and practices identified in the analysis, though based on questions regarding the teacher's perspective, include both the teachers and the learners. A learner not understanding their reused code, or not having a proper understanding of their general programming level, is a challenge important for the learner as much as it is for the teacher. Likewise, when speaking about practices such as using good exercises with many possible solutions, or focus on code modifications in programs for triggering understanding, these are practices regarding the learners learning equally much as the teachers teaching. Of course teaching and learning cannot, and should not, be considered separate from one another, which can be easily forgotten. This really emphasizes the importance of the theoretical framework in formative assessment, how formative assessment and formative practice is achieved in the interconnection between the teacher, the learners and their peers. I think even though the questions regard the teacher's perspective, reading this report from a teacher-to-learner perspective would at best be shallow, and I think similar studies taking the learner perspective could be very valuable. Making learners active participants in their own learning, their education, their goals, ways of working and general practice in school has intrinsic value in giving them autonomy, freedom and responsibility in their own learning, and evidently has remarkable learning gains for the learners (Black & William, 1998; Black, 1998; Black et al., 2004; Black & Wiliam, 2009b). In addition to, or connected with, increased learning, several studies on motivational factors in mathematics show that learner autonomy, self-determination and ownership of the learning activities are very important factors for learners' motivation (see e.g. Wæge & Nosrati, 2018)

9.3 Code Reuse and Achievement Level in Programming

In chapter 8.3.2 about achievement level in programming, there were different claims from the respondents in terms of how we should interpret the general achievement level of learners in programming in relation to code reuse. From what I can understand and interpret from the interviews, there are some connections between code reuse and general achievement level in programming. Some of the teachers claim that learners with a high level in programming also are the ones reusing code most frequently, while others point out that there can be learners very skilled in code reuse but less skilled in manual programming, and vice versa. When diving back into the statements, I think it becomes clearer to speak about *types* of code reuse in relation to general achievement level in programming. When the teachers describe the code reusing learners as most skilled, they also describe the code reuse in terms of being able to understand the fundamental structure of exercises and thus knowing what to look for online, finding the best resources in the right places, modifying and adjusting the reusable code artifacts to fit their problem, and solve programming exercises efficiently and smart with code reuse. When low achieving learners are mentioned together with code reuse as a practice, the teachers are mostly talking about straight up copy-and-paste practice, googling an exercise and copying the first thing they find and at the best adjusting variable names to Norwegian. I think teachers can assess learners to some extent in terms of the type of code reuse. Going back to the theoretical framework on code reuse in chapter 3.3, I think teachers can assess learners in terms of good code *retrievement*, *understanding and evaluation* of the code retrieved, *modifications* to fit the context and the *integration* in the new code or program. At the same time, there will also be learners who won't reuse any code, creates everything form scratch and are good programmers that way, and that of course needs to be taken into consideration.

Now, saying that high achieving learners practice code reuse in a certain way, and low achieving learners in another, doesn't mean low achieving learners are reusing code in a wrong way. Rather, they are doing it at a level suitable for them. Many of the teachers from the interviews made it clear that there are significant learning gains just from copying code, especially for the lowest achieving learners. This can be copying programs just to get a feeling of how interfaces and programs work together, and just seeing something work. It can also be that a learner doesn't understand a certain aspect of a problem, and copies code for that exact aspect, being able to deliver working solutions. Also, if a learner only understands 10% of a programming topic, they can do that 10%, then reuse or copy the 90% remaining and still deliver a working product. I think this is very important for their sense of mastering and motivation in programming education. Equally important is, as the teachers point out, that the lowest achieving learners may *learn more* from copying, than simply not understanding and being able to work with a problem. Zanders et. al in their article *Copying Can be Good: How Instructors Use Imitation in Teaching Programming*, introduce perspectives on copying, imitation and mimicry as important ways of learning programming, and how a significant amount of educators use these techniques as im-

portant steps into programming (Zander et al., 2019). There are interesting learning gains in copying, and a formative practice in programming education should consider this.

In terms of a formative view of learning and formative assessment, I think teachers also should consider the possibilities code reuse can offer in terms of learners learning parts of what they are supposed to, as described in the paragraph above, and learning the rest later. I think a sustainable, formative view on learning would welcome learners learning parts of what they are supposed to and reusing the rest, and then learning the missing parts when they are ready, have time or are motivated for it. Especially using portofolio assessment and open exercises, as some of the teachers did mention, the learners can work e.g. with a project, and instead of delivering and doing something new, they can deliver a working product, go back and revise parts they didn't understand, and iteratively gain the knowledge they are supposed to. I think code reuse enables a lot of these processes.

To sum up, learners at different achievement levels in programming seem to perform different types of code reuse, and different types of code reuse seem to be appropriate for learning programming at different achievement levels. This is an important result in terms of the three steps of formative assessment; the teacher trying to understand the current level of the learner, deciding on goals for the learner, and deciding what the learner needs to do in order to achieve those goals. The interviews suggest how learners reuse code can indicate their current level and what their goals should be, and then ways of reusing code can contribute to the learners reaching their goals in programming.

9.4 A Clear Focus on Goals

Goal attainment was not mentioned that much in the interviews, yet I think it's relevant to discuss, and that a lot of the challenges presented touches the formative work with goals when learning.

From the results, we can read that code reuse is sometimes accepted, and sometimes is not. This depends on the teacher, the types of problems and the learners. For the most restrictive teachers it seems like the learners should do online searches as a last resort, and that code reuse is accepted at the level of searching for modules and libraries in the programming language interfaces, and use built-in functions and help. For the less restrictive teachers with a more positive attitude towards code reuse, it seems like almost all code reuse is encouraged as long as the learners don't copy whole projects or too large parts of code, and that they keep the formal requirements in order, such as references and documentation.

I think that no matter how restrictive the teachers are, code reuse is an aspect of programming that requires very clear communication about what the goals are for the learning, and how it can be achieved. As exemplified a couple of times already, if sorting a list of numbers is the *goal* of an exercise, surely it can't be acceptable at a certain level to just copy code that sorts numbers. That is, if the goal is not to find

code somewhere that helps with sorting. If sorting is a sub problem, then perhaps using a built-in algorithm to sort the numbers is accepted, but in other situations the teacher might want the learners to code *everything*. No matter how you see it, there will be some level of reuse either way. Using a high level programming language such as for instance *Python* reuses solutions and software that make the computer interpret the human language commands written. Programming consists in many ways of several layers of abstraction and reuse, and I think it is highly important for the teacher to communicate which level of abstraction and reuse the learners work at, what the goals are and what they are allowed to reuse and not. In terms of individual learners, this might get even more difficult; if a teacher *knows* that one learner for certain is able to write out e.g. a sorting algorithm, maybe the teacher should encourage that learner to just reuse it and not spend too much time on it. For other learners, the teacher might need to demand "manual labor" to be sure the learners understand the concepts. This is a very complex topic, demanding precise and well-communicated goal attainment, and knowledge of each individual learner.

From the interviews, it emerges that most of the teachers work alone as programming teachers at their schools, and that there are no formal guidelines for how code reuse should be done and what should be considered acceptable code reuse. Some of the teachers also explain that they haven't had that much focus on code reuse in particular because it hasn't been such a large part of the curriculum. I think that with the coming curriculum most likely including very specific goals considering code reuse, that well thought through guidelines, goals and accepted practice will be very important for teachers to discuss. Andy Hargreaves and Michael Fullan argue that successful schools in terms of pedagogic capacity, or *better teachers*, are recognized by that teachers cooperate, that the teachers' work is visible to one another and for the school, that the teacher profession is characterized by mutual dependence as well as external accountability, and that the professional knowledge is developed and consolidated (Hargreaves & Fullan, 2014, p.107-108). Though this has to do with lack of colleagues, it can be argued that formative assessment in terms of code reuse (and programming in general) is affected by colloquial cooperation, and that a lack of collaborated frameworks or guidelines can hurt the assessment work.

To sum up, goal attainment is very important, and even though the teachers are well educated in programming, they should probably not be alone about deciding on all these questions.

9.5 Computational Thinking and Code Reuse

Reading the theoretical framework revolving around code reuse and how the teachers speak about the different skills enabling learners to perform good code reuse, it is hard not to think of *computational thinking*. Computational thinking, explained briefly, is about abstract problem solving relevant, but not limited to, programming, and involves analysing problems, breaking problems into smaller sub problems, finding abstractions, applying algorithms, tools and techniques to solve problems and the

application of high level thinking to solve problems (García-Peñalvo, Reimann, Tuul, Rees, & Jormanainen, 2016). Some promoters of computational thinking even call it *a 21st century skill*, being a skill beyond specific subjects like mathematics and computer science, and more of an integrated way of thinking and solving problems necessary for the present time (García-Peñalvo et al., 2016).

International educational research is very concerned and engaged with it (see e.g. Bocconi, Chioccariello, & Earp, 2018), as are newer Norwegian areas of priority when it comes to natural science subjects (Utdanningsdirektoratet, 2019). When the teachers describe good code reuse as learners being able to understand their problems at a high level, designing abstract solutions before solving them, breaking down problems, knowing what they look for, etc., it seems very interesting to map code reuse skills or practice to computational thinking skills or practice. I think this is very worth mentioning if one argue that the skills used for code reuse are similar to the ones for computational thinking and therefore very important, or if there are direct connections between the two, and it would certainly be an interesting topic for further research.

9.6 Open and Rich Exercises

In the results considered with types of exercises for formative assessment and code reuse, the teachers mention that the types of exercises are very important in order to assess the learners in an appropriate way. The types of exercises promoted by the teachers are those that are open in nature, usually meaning that there are many ways of solving them and that they have several correct solutions. The teachers also mention rich exercises, meaning they are open, and that learners at both low and high levels can work with the same exercise, but at different levels in terms of the quality. The way some teachers give open exercises, both enables code reuse, and makes sure that solutions cannot be just copied, because the learners need to design their own solutions and sometimes even define their own problems. It is worth mentioning that research on mathematical didactics emphasize that open and rich problems are very important and effective for learning gains and learner motivation (Karlsen, 2014; Wæge & Nosrati, 2018). I think open and rich exercises are important in programming education in general, but especially so that code reuse as a practice can be utilized appropriately. In terms of computational thinking mentioned above, code reuse can be used to realise programming practice where learners work on a high level of abstractions and can train their problem solving skills, rather than programming techniques. That is, if that is a desirable goal.

9.7 General Formative Practice and Challenges

Though presenting challenges, many of the teachers made clear that when working formative and continuously with assessment, many of the challenges were non-existing. This emphasizes the importance of a general formative practice, and that formative assessment should be included in every part of the learning, as discussed in chapter

3.2 on formative practice. Another important aspect is that formative assessment is not only about evaluating every solution to exercises, but a multi dimensional practice as shown in the results, assessing through conversations, discussions, learners documentation and comments in code, code modifications and presentations.

However, some of these practices can be time consuming, and it is worth discussing. Most of the teachers in the interviews had fairly small classes, from six to around 15 learners in their class. With bigger classes, many of these types of practices can be less feasible to implement. On the other hand, promoters of formative assessment as a general practice claims not only that it has gains for learning, but is also in the long run a more efficient way of working (Black, 1998). Also, for the challenge of the teacher not being able to understand what the learners are doing, in case of their reused code, a solution would be being part of their process, do continuous assessment and ask the learners to present code in different formats in order to mitigate these difficulties. There is likely a trade-off between performing a multitude of types of assessment, and the time and resources used. Teachers should be aware of their possibilities, their restrictions and how they can work to promote each learners learning. Continuous assessment or multi dimensional assessment are not silver bullets to perform good teaching, but are concepts and ideas to follow and implement in suitable ways to increase learning.

9.8 Plagiarism and Cheating

A particularly interesting thing about the results is that plagiarism has not been as much of a topic of discussion as I expected. For the pilot project described in chapter 2.2 on assessment in programming education in general, there were a lot of talk about plagiarism, and only a little in this project. I interpret this as that plagiarism and cheating becomes less important when speaking of formative assessment, i.e. assessment only with the intention of creating positive changes for learning. Of course, the teachers in the interviews were touching the problem when explaining challenges around learners reusing code they don't understand, which in a formal or legal point of view would be considered plagiarism or cheating in some cases. However, with the sole purpose of learning, it becomes less relevant. This also applies to the statements from the teachers about creating exercises in which the solutions cannot be easily copied or retrieved with online searches. Even so, many of the teachers claimed that if the learners copied, the teachers themselves and their choice of exercises have to take some of the blame.

Nevertheless, all classroom practice is interconnected, and it is not possible to also consider summative assessment, fair grading, intellectual products etc., when speaking of code reuse. However, when only considering learning gains, it becomes less important. I think a lot of the practices, such as opens exercises, multi dimensional assessment and continuous assessment can be utilized to discover cheating, but in terms of formative practice it is used to increase learning. Either way, the practice is appropriate and useful, and one intention does not necessarily need to exclude the

other. If learners are not able to copy solutions directly and need to design solutions, break problems into sub problems and write code themselves, this is probably good for pure learning gains, *and* to avoid cheating or plagiarism. The teachers, e.g. in T5's quote in 8.1.4 on learners finding solutions to exercises on YouTube, is clearly also concerned with cheating as well as the lack of learning.

9.9 Knowledge Reuse in Programming and Other Subjects in School

An interesting aspect to discuss is whether the challenges and suggested practices are in any way unique for programming, or if the reuse perspective is applicable to other subjects as well. One of the things I have identified as perhaps most important for allowing the challenges to emerge in the first place, is that with code reuse in programming, you are able to solve problems at a certain level without necessarily having knowledge in a former level or step leading up to it. As mentioned in the discussion part 9.1 on attitudes towards code reuse, there is an approach to learning which can be seen as deductive or with an axiomatic logic, where there are logical, linear steps or building blocks to master before moving on to the next. With this approach, learners are supposed to learn a step A before they move on to B. Then they can (re)use A and B in order to learn C. However, in programming, it is easier to skip steps without having learned everything bottom-up. You do not necessarily need to learn every lower level to work at a higher level. Two interesting questions are if this is unique to programming, and whether it is a problem or not.

As for the first question, I would argue that being able to skip logical or deductive steps is not unique for programming, but is perhaps easier to utilize or more visible in a sense. For instance, in mathematics with a logical deductive approach, it could be expected that the learner first learns the definition of the derivative, and the basic concepts of differentiation, before working with problems of derivatives and slopes of functions. However, if the learner has no understood the definition of a derivative, they can still continue with differentiating functions by rules or with software, and work with mathematical aspects related to it. For social sciences, e.g. sociology, it may be expected that a learner can define, explain and provide examples of what *socialization* is, before using the term to discuss or understand other concepts. Or in a paper, a learner could write that some group has a *social identity* perhaps without knowing what identity means, if the reader takes it for granted that the writer knows the term. At some point the learners will be at a level where they use the terms, without anyone checking if they know the basics. I would therefore, in some sense, argue that no, the fact that one can skip learning goals and levels because of reuse is not unique for programming. However, the implication that you can create complex working programs is perhaps more seemingly astonishing or spectacular in some sense. However, fully understanding and discussing this is beyond the scope of this project.

As for the second question regarding whether it is a problem or not that the learners can skip levels, I think this depends on who it becomes a problem for. If learners

are supposed to be graded fairly with the same starting points, it can be a problem if some learners reuse code and make seemingly better products and thus unfairly receive better grades than others. In terms of learning, i.e. a formative approach, it can be a problem if a learner seemingly knows and understands something, but have not learned. This can be both the teacher not catching this, or the learner themself not having a proper understanding of their own achievement. On the other hand, code reuse can allow learners to work with more motivating, more interesting and more fun exercises and projects, and there is no standard stating that one amount of learning is better or worse than one amount of motivation and joy in producing programs. Nor is *learning* easy to define. I think the most important aspect here is that teachers are clear and communicate goals well, so that the learners can work at their level or above, but have a clear understanding of what they are creating themselves, what they are reusing, what they understand and not in terms of programming, and what is straight up reused. At this level of programming, everything could be considered reuse or at a high abstraction level, and it can be useful to know what is what.

Heading back to the discussion of whether learning should be done linearly and deductive in logical steps or more compound, I think an important point to make is that programming might be taught as it is by using motivating software, exciting interfaces and relevant examples, because it is a new academic discipline. This discussion is very abstract and hypothetical, but I believe that if mathematics, or mathematics education or didactics, was invented today, we would for instance first have the learners make games with animations where a character jumps, and then discuss how quadratic functions mathematically can simulate jumping. Perhaps the learners would first play board and card games, and then guess and discuss statistically how they could win and learn probability. Or maybe we would track running or driving patterns and then discuss velocity changes. There are likely also several teachers who promote teaching this way already. I think many subjects in school are taught with a strictly deductive learning paradigm, and that programming and the utilization of code reuse is done how it is because it is more contemporary.

10 Conclusions

I have in this project performed a grounded theory study based on the question

What are teachers' experiences with formative assessment with regard to code reuse in programming education?

The first conclusion to point out is that code reuse in programming education is a topic worth investigating further. It seems to be an important part of everyday programming practice, yet a complex topic in terms of teaching, education and, as this study points out, formative assessment. Teachers, aware of the topic or not, have several interesting experiences with code reuse in their classrooms, and there can be important gains for the future of programming education to discover research more.

RQ1: What are teachers attitudes towards code reuse in programming education, and to how is it implemented in class?

Teachers have different attitudes, awareness levels and approaches to code reuse in the classroom. Some teachers explicitly focus on teaching code reuse to the learners in an appropriate, conscious manner. Most of the teachers encourage code reuse and see it as a natural part of programming, but do not explicitly focus much on how it is done or to teach the learners how to reuse code properly. Rather, they consider programming as a whole, and guide learners through their programming, and code reuse, with their skills and knowledge. Some teachers are also more restrictive towards code reuse, given their approach towards what they wish to teach the learners and which skill sets they want the learners to attain.

RQ2: What are the challenges for the teacher when working with formative assessment regarding code reuse as a practice when programming?

Code reuse as a programming practice has some consequences in terms of challenges that the teachers experience when doing formative assessment in programming classes.

One of the challenges is that learners reuse code which they do not understand. This is a challenge when the teacher is to assess learners work and guide them onward, and it can hurt the learner's self-assessment and understanding of their own achievement level.

The amount and type of code reuse seems to have some indications as to the general programming achievement level of the learners, and at the same time be misleading. This makes understanding the general achievement level of the learners challenging.

Sometimes code reuse as a practice can happen on the expense of general or specific programming training. This means that the learners reuse code in order to work efficiently, but that they perhaps in the long run will lack general manual programming skills. When reusing code for specific tasks the learners also might not

learn the specific skill to be practiced. Teachers see this as challenging when trying to decide what the goals of the learners are and what the next steps in their learning should be.

And as a last challenge, when learners reuse very advanced code to create complex, advanced and large programs, it can either be impossible or in terms of time infeasible for the teachers to guide the learners further or help them if they are stuck. Code reuse allows the learners to work with perhaps more fun, motivating and complex examples, but the reused code can become too advanced to be appropriate for learning programming.

RQ3: What are some best practices for conducting formative assessment from a teacher's perspective, regarding code reuse as a practice when programming?

To mitigate or overcome these challenges, and to appropriately work with formative assessment in programming, the teachers from the study suggest several best practices.

Tasks and exercises should be designed with awareness for code reuse and for formative use; open exercises with many possible solutions and ways of achieving them, rich exercises which are easy to understand but have some difficult aspects, exercises where learners need to design their solution and break down problems, and exercises where code reuse can be a natural part of the solution, but not *the* solution.

Formative assessment should be considered something teachers do in all parts of classes, not just when evaluating exercises, and they should work with continuous assessment in order to keep track of the current level, challenges and goals for each individual learner.

As code reuse seems to affect assessment as well as other parts of the programming classes, teachers need to conduct formative assessment of the code reuse in particular; guide the learners to be aware of how code reuse is done, what the formal requirements are in terms of documentation and referencing, and *teach* and assess the code reuse as well as the programming in general.

A particularly efficient method of gaining understanding of achievement level and the particular code reuse, both for the teacher and the learner, is testing the code with different code modifications. Code reuse in theory is concerned with evaluating code, modifying it to fit new contexts and integrating it in own programs, and a particularly efficient way of moving code reuse from copying to appropriate code reuse is to do, test and practice code modifications.

The final practice is concerned with being aware of the possible ways of assessing necessary, and performing formative assessment with several different dimensions, attributes of variables; learners presenting code with reports, documentations, code comments, oral presentations and video presentations, asking learners to modify code, teachers and learners testing programs and running through code together, peers using each others' programs an running evaluations of different parts and of different dimensions of the learners' programs.

10.1 Further Research

Code reuse in programming education is definitely an interesting topic, and there are more issues to research.

Though identifying challenges and practice for formative assessment in this study, this is done in an exploratory manner, and the results should be tested, verified and specifically researched in more depth. For instance, it would be very interesting to conduct more research into what types of exercises are effective and not for different goals. For programming subjects where one can expect that many solutions to exercises already exists, such as introductory programming or algorithms and data structure courses, it would be interesting to find out more about how instructors and teachers set up practice exercises to avoid copying and mindless reuse. Also finding more attributes regarding types of code reuse, compared with general skill level in programming, would be very interesting.

This research study was done from a teacher perspective. Research on formative assessment, as well as didactics in general, emphasizes the importance of the learner as an active agent in the all parts of the learning. It would be particularly interesting to investigate these topics from a learner's perspective as well.

Given that teachers in this small scale study had different attitudes towards code reuse, it would be interesting to find out more about what teachers think of this topic. Given that many of the teachers from this study worked alone with teaching programming, we should find out more about how teachers in larger collegiate groups work with this.

And further, researching code reuse not only in a business scope discussing cost effectiveness and saving time, but in terms of efficient, interesting and motivating work in programming education is an important topic to investigate further. Especially with contemporary attitudes towards computational thinking including skills related to abstraction, problem design and problem solving, I think code reuse has an important part to play in programming education.

References

Agresti, W. (2011). Software reuse: Developers' experiences and perceptions. *Journal of Software Engineering and Applications*, *4*(1), 48–48. Retrieved from http://search.proquest.com/docview/889406715/

Alfasoft. (n.d.). *Nvivo.* Retrieved from https://www.alfasoft.com/en/products/statistics-and-analysis/nvivo.html

Ayala, C., Franch, X., Conradi, R., Li, J., & Cruzes, D. (2013). Developing software with open source software components. In S. E. Sim & R. E. Gallardo-Valencia (Eds.), *Finding source code on the web for remix and reuse* (1st ed. 2013. ed., p. 167-186). New York, NY: Springer New York : Imprint: Springer.

Black, P. (1998). *Inside the black box : raising standards through classroom assessment.* London: GL Assessment.

Black, P., Harrison, C., Lee, C., Marshall, B., & Wiliam, D. (2004). Working inside the black box: Assessment for learning in the classroom. *Phi Delta Kappan Magazine*, *86*(1), 8–21.

Black, P., & Wiliam, D. (2009a). Aspects of formative assessment [illustration] developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education)*, *21*(1), 8.

Black, P., & Wiliam, D. (2009b). Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education)*, *21*(1), 5–31.

Black, P., & William, D. (1998). Assessment and classroom learning. *Assessment in Education: Principles, Policy Practice*, *5*(1), 7–74. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/0969595980050102

Bocconi, S., Chioccariello, A., & Earp, J. (2018). *The nordic approach to introducing computational thinking and programming in compulsory education. report prepared for the nordic@bett2018 steering group.* National Research Council of Italy, Institute for Educational Technology (CNR-ITD). Retrieved from "https://doi.org/10.17471/54007"

Brombach, H. (2020a). digi.no: Bruker du zoom til videomøter? da bør du vite dette. Retrieved from https://www.digi.no/artikler/bruker-du-zoom-til-videomoter-da-bor-du-vite-dette/489071

Brombach, H. (2020b). digi.no: Uninett kjører zoom på nordiske servere. Retrieved from https://www.digi.no/artikler/uninett-kjorer-zoom-pa-nordiske-servere/489201?key=cJm78XYm

Carlsen, H. (2020). Nrk: Forbyr ansatte å bruke populær videokonferansetjeneste. Retrieved from https://www.nrk.no/norge/forbyr-ansatte-a-bruke-populaer-videokonferanse-tjeneste-1.14968765

Engelsen, B. U. (2006a). *The didactic relational model [illustration], in kan læring planlegges? : arbeid med læreplaner - hva, hvordan, hvorfor* (5. utg. ed.). Oslo: Gyldendal akademisk.

Engelsen, B. U. (2006b). *Kan læring planlegges? : arbeid med læreplaner - hva, hvordan, hvorfor* (5. utg. ed.). Oslo: Gyldendal akademisk.

Engh, K. R. (2017). *Vurdering for læring i skolen : på vei mot en bærekraftig vurderingskultur.* Oslo: CAPPELEN DAMM.

Frakes, W., & Kang, K. (2005). Software reuse research: status and future. *IEEE Transactions on Software Engineering*, *31*(7), 529–536.

Gallardo-Valencia, R. E., & Sim, S. E. (2013). Software problems that motivate web searches. In S. E. Sim & R. E. Gallardo-Valencia (Eds.), *Finding source code on the web for remix and reuse* (1st ed. 2013. ed., p. 253-270). New York, NY: Springer New York : Imprint: Springer.

García-Peñalvo, F. J., Reimann, D., Tuul, M., Rees, A., & Jormanainen, I. (2016). *An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers.* Zenodo.

GDPR. (n.d.). *Gdpr - general data protection regulation.* Retrieved from `https://gdpr-info.eu/`

Haefliger, S., von Krogh, G., & Spaeth, S. (2008). Code reuse in open source software.(report). *Management Science*, *54*(1), 180.

Hargreaves, A., & Fullan, M. (2014). Profesjonell kapital. In *Arbeidskultur for bedre læring i skolen* (p. 99-122). Oslo: Kommuneforlaget.

Icsr. (n.d.). Retrieved from `https://link.springer.com/conference/icsr`

ILU. (n.d.). *Ntnu institutt for lærerutdanning: Ressurser om vurdering for læring.* Retrieved from `https://www.ntnu.no/ilu/ressurser-vurdering-laring`

Karlsen, L. (2014). *Tenk det! : utforsking, forståelse og samarbeid - elever som tenker sjæl i matematikk : ungdomstrinnet.* Oslo: Cappelen Damm akademisk.

Kvale, S., & Brinkmann, S. (2015). *Det kvalitative forskningsintervju* (3. utg. ed.). Oslo: Gyldendal akademisk.

lovdata.no. (1998). *Lov om grunnskolen og den vidaregåande opplæringa (opplæringslova).* Retrieved from `https://lovdata.no/dokument/NL/lov/1998-07-17-61`

lovdata.no. (2009). *Forskrift til opplæringslova kapittel 3. individuell vurdering i grunnskolen og i vidaregåande opplæring.* Retrieved from `https://lovdata.no/dokument/SF/forskrift/2006-06-23-724/KAPITTEL_4#KAPITTEL_4`

Lær kidsa koding. (n.d.). Retrieved from `https://www.kidsakoder.no/`

Microsoft. (n.d.). *Microsoft teams.* Retrieved from `https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software`

Nakakoji, K., Yamamoto, Y., & Nishinaka, Y. (2013). Unweaving code search toward remixing-centered programming support. In S. E. Sim & R. E. Gallardo-Valencia (Eds.), *Finding source code on the web for remix and reuse* (1st ed. 2013. ed., p. 17-34). New York, NY: Springer New York : Imprint: Springer.

NESH. (2015). *The norwegian national committees for research ethics - about.* Retrieved from `https://www.etikkom.no/en/our-work/about-us/the-national-committee-for-research-ethics-in-the-social-sciences-and-the-humanities-nesh/about-nesh/`

NESH. (2016). *Guidelines for research ethics in the social sciences, humanities, law and theology* (4th ed. ed.). Oslo: The National Committee for Research Ethics in the Social Sciences and the Humanities. Retrieved from https://www.etikkom.no/globalassets/documents/english-publications/60127_fek_guidelines_nesh_digital_corr.pdf

NSD. (n.d.-a). *Nsd - norwegian centre for research data.* Retrieved from https://nsd.no/nsd/english/index.html

NSD. (n.d.-b). *Nsd - verktøy for datahåndtering.* Retrieved from https://nsd.no/datahandtering/

NTNU. (n.d.-a). *Behandle personopplysninger i student- og forskningsprosjekt.* Retrieved from https://innsida.ntnu.no/wiki/-/wiki/Norsk/Behandle+personopplysninger+i+student-+og+forskningsprosjekt

NTNU. (n.d.-b). *Ntnu - collection of personal data for research projects.* Retrieved from https://innsida.ntnu.no/wiki/-/wiki/English/Collection+of+personal+data+for+research+projects

NTNU. (n.d.-c). *Ntnu - zoom videoundervisning.* Retrieved from https://innsida.ntnu.no/wiki/-/wiki/Norsk/Zoom+videoundervisning

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

Sevik, K. (2016). Programmering i skolen. Retrieved from https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Sim, S. E., & Gallardo-Valencia, R. E. (2013). Introduction: Remixing snippets and reusing components. In S. E. Sim & R. E. Gallardo-Valencia (Eds.), *Finding source code on the web for remix and reuse* (1st ed. 2013. ed., p. 1-14). New York, NY: Springer New York : Imprint: Springer.

Sojer, M. (2011). *Reusing open source code.* Springer Verlag.

Umarji, M., & Sim, S. E. (2013). Archetypal internet-scale source code searching. In S. E. Sim & R. E. Gallardo-Valencia (Eds.), *Finding source code on the web for remix and reuse* (1st ed. 2013. ed., p. 35-52). New York, NY: Springer New York : Imprint: Springer.

Uninett. (n.d.-a). *Uninett as website.* Retrieved from https://www.uninett.no/en

Uninett. (n.d.-b). *Uninett zoom: Facts concerning gdpr og privacy.* Retrieved from https://www.uninett.no/en/uninett-zoom-facts-concerning-gdpr-og-privacy

Utdanningsdirektoratet. (n.d.-a). *Fagfornyelsen.* Retrieved from https://www.udir.no/fagfornyelsen

Utdanningsdirektoratet. (n.d.-b). *Læreplan i informasjonsteknologi - programfag i utdanningsprogram for studiespesialisering (inf1-01).* Retrieved from https://www.udir.no/kl06/INF1-01

Utdanningsdirektoratet. (n.d.-c). *Læreplan i informasjonsteknologi - programfag i utdanningsprogram for studiespesialisering (inf1-01) - informasjonsteknologi 2.* Retrieved from https://www.udir.no/kl06/INF1-01/Hele/Kompetansemaal/informasjonsteknologi-2

Utdanningsdirektoratet. (2016). *Forsøkslæreplan i valgfag programmering.* Retrieved from https://www.udir.no/kl06/prg1-01

Utdanningsdirektoratet. (2017). *Forsøkslæreplan i programmering og modellering x - programfag i utdanningsprogram for studiespesialisering.* Retrieved from https://www.udir.no/kl06/PRM1-01

Utdanningsdirektoratet. (2019). *Algoritmisk tenkning.* Retrieved from https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020a). *Læreplan i informasjonsteknologi (utkast).* Retrieved from https://hoering.udir.no/Hoering/v2/962

Utdanningsdirektoratet. (2020b). *Udir.* Retrieved from https://www.udir.no/in-english/

Whereby. (n.d.). *Whereby.* Retrieved from https://whereby.com/

Wæge, K., & Nosrati, M. (2018). *Motivasjon i matematikk.* Oslo: Universitetsforl.

Zander, C., Eckerdal, A., McCartney, R., Mostrom, J. E., Sanders, K., & Thomas, L. (2019). Copying can be good: How instructors use imitation in teaching programming. In *Proceedings of the 2019 acm conference on innovation and technology in computer science education* (p. 450–456). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3304221.3319783 doi: 10.1145/3304221.3319783

Zoom. (n.d.). *Zoom - website.* Retrieved from https://zoom.us/

A Appendix: Information Note

Vil du delta i forskningsprosjektet "Formative assessment of learners' code reuse in programming education"?

Til lærere i programmering i videregående skole

Jeg er student på lektorprogrammet i realfag ved NTNU i Trondheim og skal gjennomføre et forskningsprosjekt i forbindelse med min masteravhandling om programmeringsundervisning. I dette skrivet gir jeg deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Formålet med prosjektet er å få innsyn i læreres erfaringer, tanker og refleksjoner rundt vurderingsarbeid med hensyn på kodegjenbruk i programmering, og å kunne avdekke og drøfte utfordringer knyttet til dette. Fordi programmering vil innføres med fagfornyelsen i flere fag, og vurdering er et omdiskutert tema skolen, vil det være hensiktsmessig å få mer kunnskap om dette fra lærere med erfaring innen programmeringsundervisning og -vurdering.

Konkret ønsker jeg å få innsyn i erfaringer i den delen av undervisnings- og vurderingsarbeid som omfatter formativ vurdering, altså vurdering med hensikt å skape positive forandringer i det videre læringsarbeidet for elevene. Problemstillingen for prosjektet lyder "What are teachers' experiences with formative assessment of learners with regard to code reuse in programming education?". På norsk: "Hva er læreres erfaringer med formativ vurdering av elever med hensyn på kodegjenbruk i programmeringsundervisning?"

Resultatene av studien vil bli brukt i mitt masterprosjekt.

Hvem er ansvarlig for forskningsprosjektet?

NTNU, Institutt for Datateknologi og Informatikk (v/Monica Divitini) er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?

Fordi du har erfaring med å undervise og vurdere programmering hos elever i norsk skole.

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, innebærer det at du deltar i et personlig intervju med meg som varer ca. 30-45 minutter. Du vil bli spurt om din erfaring med å undervise og vurdere programmering og kodegjenbruk. Opplysningene vil registreres med lydopptak som vil transkriberes slik at alle personlige opplysninger eller opplysninger som kan bidra til å gjenkjenne deg vil utelates. På grunn av omstendighetene med

Covid-19 vil intervjuene gjennomføres over skype eller lignende tjeneste som tilbyr ende-til-ende kryptering på samtaler, men det vil benyttes en ekstern lydopptaker som verken er koblet til datamaskinen eller internett, slik at opplysningene er trygge. Kun lyd vil tas opp.

Det er frivillig å delta

Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykke tilbake uten å oppgi noen grunn. Alle opplysninger om deg vil da bli anonymisert. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger

Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket. Kun jeg og veileder vil ha tilgang til opplysningene. Navnet og kontaktopplysningene dine vil erstattes med en kode som lagres på en egen navneliste adskilt fra øvrige data. Datamaterialet vil lagres på en kryptert minnepinne som vil være innelåst når den ikke er i bruk. Du vil ikke kunne gjenkjennes i publikasjon.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?

Prosjektet skal etter planen avsluttes juli 2020. Personopplysninger og opptak vil slettes ved prosjektslutt.

Dine rettigheter

Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om deg,
- å få rettet personopplysninger om deg,
- få slettet personopplysninger om deg,
- få utlevert en kopi av dine personopplysninger (dataportabilitet), og
- å sende klage til personvernombudet eller Datatilsynet om behandlingen av dine personopplysninger.

Hva gir oss rett til å behandle personopplysninger om deg?

Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra NTNU, institutt for datateknologi og informatikk, har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?

Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med:

- Niklas Nystad, 97507076, niklasjn@stud.ntnu.no
- NTNU, institutt for datateknologi og informatikk v/Monica Divitini, 91897790, divitini@ntnu.no
- Vårt personvernombud: Thomas Helgesen.
- NSD – Norsk senter for forskningsdata AS, på epost (personverntjenester@nsd.no) eller telefon: 55 58 21 17.

Med vennlig hilsen
Prosjektansvarlig Monica Divitini
Student Niklas Joakim Nystad

B Appendix: Definitions of Terms to Respondents

Formativ vurdering

Definisjon: vurderingsaktiviteter eller vurderingspraksis som utelukkende har til hensikt å skape positive forandringer i det videre læringsarbeidet. I kontrast til summativ vurdering som har til hensikt å uttrykke nåværende kompetanse uten å benytte dette videre. Generelt kan vi si at vurderingspraksis blir formativ når det gjøres for å oppnå bedre læringsutbytte hos elevene. Vurdering for læring.

Prosjektet benytter en bred forståelse av vurdering, hvor det kan forstås som alt fra formelle eksamener og prøver, til klasseromsdialoger, observasjon av elever, enkeltsamtaler eller selvvurdering hos elevene selv.

Kodegjenbruk

Definisjon: Prosessen med å benytte eksisterende kode eller kunnskap om kode til å skape eller utvikle ny kode, programmer eller programvare, heller enn å utvikle fra bunnen av.

Prosjektet benytter en bred forståelse av eksisterende kode eller kunnskap om kode, hvor det kan forstås som kodelinjer elevene finner på nett eller i bøker, funksjoner, biblioteker, komponenter eller annet gjenbrukbart kodemateriale. Gjenbruken kan blant annet omfatte ren kopiering av kodelinjer, kopiering av strukturer med modifikasjoner og bruk av funksjoner, komponenter og biblioteker.

Kodegjenbruk i klasserommet

To perspektiver i prosjektet:

1. Kodegjenbruk som et konsept elevene skal lære
2. Kodegjenbruk som en praksis elevene gjør når de programmerer

C Appendix: Interview Guide

Formelt før intervju- og opptaksstart

- Formålet med undersøkelsen
- Fortelle hvordan intervjuet dokumenteres og bearbeides
- Anonymisering
- Taushetsplikt
- Samtykke
- Antyde hvor lenge intervjuet varer
- Informere om rett til å unnlate å svare på spørsmål, eller avbryte intervjuet
- Informere om rett til å trekke seg, også lenge etter at intervjuet er ferdig

Definisjoner og avklaringer før intervju- og opptaksstart

- Ute etter dine erfaringer, refleksjoner, tanker. Beskriv gjerne nøye.
- Formativ vurdering [i begrepsdokumentet]
- Kodegjenbruk [i begrepsdokumentet]
- Temaet er snevert, og det kan hende alle spørsmålene ikke alltid gir mening eller er like lette å svare på, men det er også viktige resultater.
- Har du noen spørsmål til temaet, eller noe annet før vi starter?
- Jeg kommer til å ta notater

Introspørsmål. Komme i gang

- Hvor mange år har du arbeidet som lærer?
- Hvor mange år har du vært lærer på den aktuelle skolen?
- Hvilke trinn underviser du på?
- Hvilke fag, kurs eller emner underviser du?
- Hvor lenge har du undervist programmering?

Hovedspørsmål

På hvilke måter jobber du med formativ vurdering eller vurdering for læring i programmering generelt?

Kan du gi meg noen eksempler på kodegjenbruk hos dine elever?

- Der elever gjenbruker kode som praksis
- Har du fokus på å lære bort gjenbruk av kode når du underviser programmering?
- Der du lærer bort, eller oppfordrer til kodegjenbruk

Hvilke fordeler og ulemper ser du med kodegjenbruk i en vurderingssituasjon?

- Fordeler/ulemper for elevene?
- Fordeler/ulemper som lærer?

På hvilke måter vurderer du elevene når det kommer til kodegjenbruk?

- Eksempler, beskrivelser?
- Erfaringer?
- Fordeler/ulemper
- Utfordringer?
- Basert på din erfaring, hvordan burde lærere jobbe med dette? Hva fungerer?
- Kodegjenbruk som praksis?
- Kodegjenbruk som noe elevene skal lære?

La oss tenke at en del av vurderingsarbeidet er å prøve å forstå hvor mye elevene kan, eller hvor de befinner seg i læringsarbeidet. Kartlegge kompetanse og forståelse. Hvordan jobber du med dette i programmeringsundervisning?

Kan det her være noen spesielle utfordringer med hensyn på kodegjenbruk?

- Eksempler, beskrivelser?
- Erfaringer?
- Utfordringer?
- Basert på din erfaring, hvordan burde lærere jobbe med dette? Hva fungerer?
- Kodegjenbruk som praksis / konsept elevene skal lære?

La oss tenke at en del av vurderingsarbeidet er å sette mål og delmål som elevene kan jobbe mot, og som passer deres kompetanse. Hvordan jobber du med dette i programmeringsundervisning?

Tar du hensyn til at elevene gjenbruker kode de f.eks. finner på nett når du setter mål for elevene?

- Eksempler, beskrivelser?
- Erfaringer?
- Utfordringer?
- Basert på din erfaring, hvordan burde lærere jobbe med dette? Hva fungerer?
- Kodegjenbruk som praksis / konsept elevene skal lære?

Tenk deg at du skal avgjøre hva elevene må gjøre for å nå målene sine og å lære, og også gi tilbakemelding på dette. Altså hvordan komme dit de skal. Hvordan jobber du med dette i programmeringsundervisning?

Påvirker kodegjenbruk arbeidet med å avgjøre hva elevene må gjøre for å nå målene sine, eller gi tilbakemelding til elevene på læringen deres?

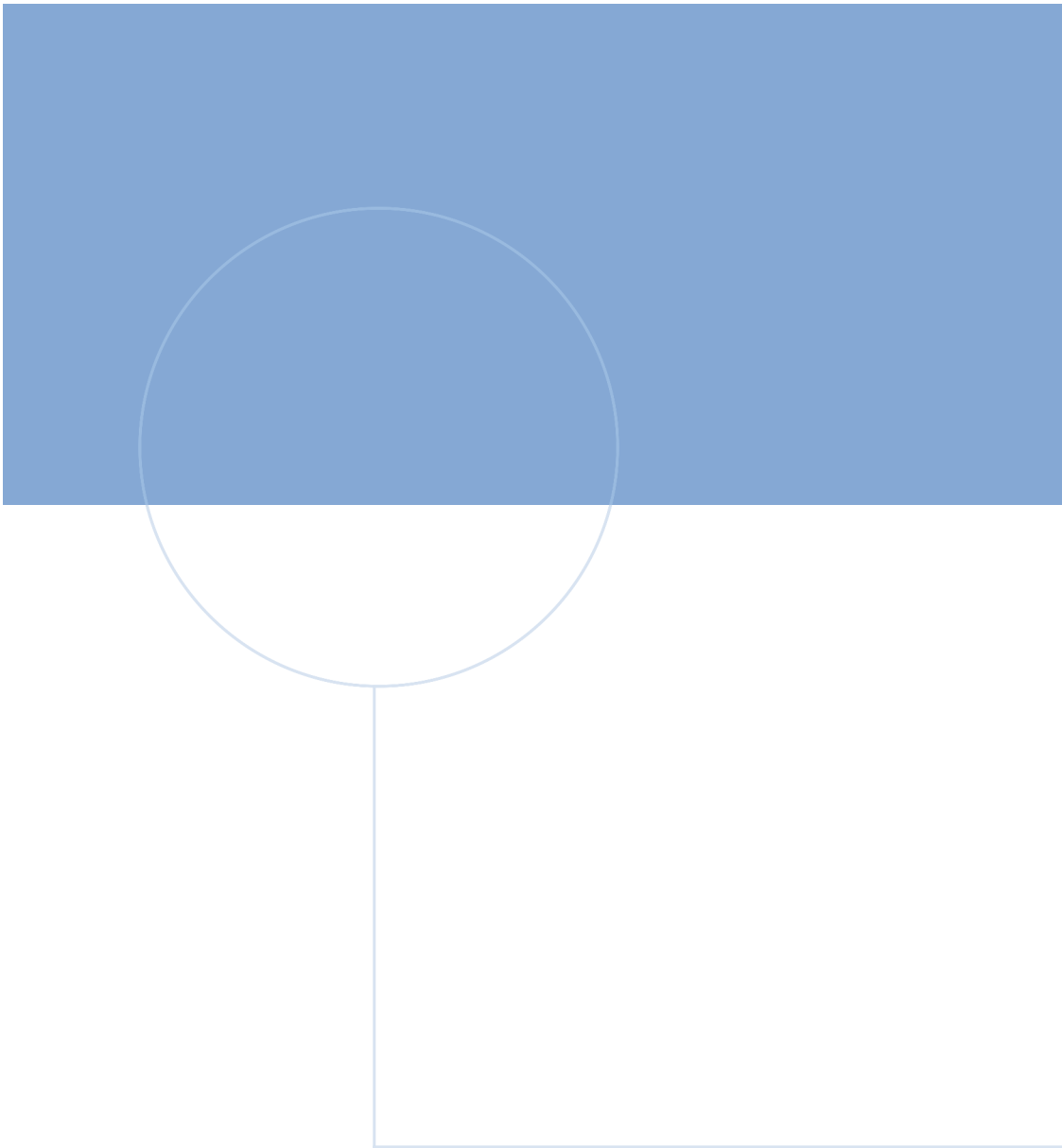
- Eksempler, beskrivelser?
- Erfaringer?
- Utfordringer?
- Basert på din erfaring, hvordan burde lærere jobbe med dette? Hva fungerer?
- Kodegjenbruk som praksis / konsept elevene skal lære?

Avslutte hoveddel

- Arbeider dere med noen av de tingene vi har snakket om på skolen du jobber på?
- Har du eller dere retningslinjer for hva som er OK gjenbruk og hva som ikke er det?
- Føler du at det kunne vært hjelpsomt med retningslinjer, treningsmateriale eller lignende for å arbeide med formativ vurdering eller kodegjenbruk?
  - Hvordan kunne det sett ut?

Avslutning

- Nå har vi jo snakket en del om temaet. Har du andre erfaringer, tanker eller refleksjoner når det kommer til dette med formativ vurdering, eller vurdering for læring, når vi snakker om kodegjenbruk?
- Er det noe jeg burde ha spurt om, eller noe du føler det ikke har vært rom for å snakke om?



NTNU

Norwegian University of
Science and Technology