

Torgeir Sandnes Laurvik

Design process behind an educational review system for student submissions

Applying knowledge from professional code reviews to an educational assessment setting

Master's thesis in MLREAL

Supervisor: Hallvard Trætteberg

December 2021



Norwegian University of
Science and Technology

Torgeir Sandnes Laurvik

Design process behind an educational review system for student submissions

Applying knowledge from professional code reviews to an educational assessment setting

Master's thesis in MLREAL
Supervisor: Hallvard Trætteberg
December 2021

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



NTNU

Kunnskap for en bedre verden

DEPARTMENT OF COMPUTER SCIENCE

MASTER'S THESIS

Design process behind an educational review system for student submissions

Applying knowledge from professional code reviews to an
educational assessment setting

Author: Torgeir Sandnes Laurvik

Supervisor: Hallvard Trætteberg

December 2021

Table of Contents

List of Figures	v
1 Introduction	3
1.1 Thesis Question	4
1.2 Target Courses	5
1.3 Definition of code review	5
2 Background	7
2.1 The courses	7
2.1.1 Introduction Course to Programming	7
2.1.2 Object-Oriented Programming course	8
2.1.3 Software and Web Development Courses	8
2.2 Formative and Summative assessments	9
2.2.1 Formative assessment (assessment for learning)	9
2.2.2 Summative assessment (assessment for control)	10
2.2.3 Assessment types in our use case	10
2.2.4 The reviewer in formative and summative assessments	11
2.3 Current system for reviewing	12
2.3.1 Introduction Course to Programming	12
2.3.2 Object-Oriented Programming	12

2.3.3	Software Development	13
2.3.4	Web Development	14
2.4	Value of Code Review-technique exposure for students	16
2.5	Motivation behind industrial CRs	16
2.5.1	Motivation in Microsoft behind doing CRs	17
2.5.2	Actual use of Code Reviews in the industry	20
2.6	Industrial Code Reviews and Our Use Case	21
2.6.1	Motivations and Objectives	21
2.6.2	Source-code in review	22
2.7	Features from industry tools	23
2.7.1	GitHub	23
2.7.2	GitLab	25
2.7.3	Collaborator	26
2.7.4	Bitbucket	29
2.7.5	Review Board	30
2.7.6	CodeFlow	31
2.8	Features from educational tools	33
2.8.1	CodePost	33
2.8.2	HTML-report generated based on tests - currently in use	34
2.8.3	CodeTour	35
2.9	Why are there differences between the educational and professional tools?	38
3	Feature Selection	40
3.1	Priority of courses	40
3.1.1	Web development	40

3.1.2	Introduction courses (ITGK and OOP) and Software development course	41
3.2	Industry-specific features	42
3.3	Selection criteria for features	42
3.4	Discussion of each feature	43
3.4.1	”Enable multiple reviewers to collaborate on the review” . . .	43
3.4.2	”Write comments in the same view as the code is inspected” .	44
3.4.3	”Streamline the review process”	45
4	Implementation considerations	49
4.1	Proposal for sprints	53
4.1.1	Sprint 1	53
4.1.2	Sprint 2	54
4.1.3	Sprint 3	54
4.1.4	Subsequent sprints	54
5	Discussion	55
5.1	Contribution	55
5.2	Limitations	56
5.2.1	Data base	56
5.3	Implications and future direction	56
	Bibliography	58
	Appendices	61
A	Initial e-mail to the lecturers	61
B	C++ email	62

C	Web development email	63
D	TDT4110 exercises description	65
E	TDT4110, exam 2020	66
F	TDT4100, exam 2017	91
G	TDT4102, exam 2019	97
H	IT2810, exam 2018	105

List of Figures

2.1	Developers' motivations for CR (Bird and Bacchelli, 2013, Figure 3)	17
2.2	GitHub: Source code changes presented in a split view	24
2.3	GitHub: Source code changes presented in a unified view	24
2.4	GitHub: Code quality annotations in review view (Github, 2021)	25
2.5	GitLab: CR interface	27
2.6	GitLab: Code quality indicators inside the review interface	27
2.7	Collaborator: CR interface	27
2.8	Collaborator: Review interface for PDF-documentation file	28
2.9	Bitbucket: Source code view	29
2.10	Bitbucket: Merge conditions	30
2.11	Review Board: Review interface	31
2.12	CodeFlow: Code quality feedback showing up in the GitHub pipeline	31
2.13	CodeFlow: Code quality overview at their web page	32
2.14	CodePost: Review interface	33
2.15	Google Docs: Comment feature	34
2.16	CodePost: De-emphasized template code	35
2.17	HTML-tool: Student written code and test results	36
2.18	CodeTour with added review comments	36

2.19	Overview of features from the studied tools	37
4.1	Mock-up: Showing the results from unit tests	50
4.2	Mock-up: The annotation from the static code analysis tool on hover	51
4.3	Mock-up: Toggle to write comments view	52

Abstract

Senior review resources for student submissions are limited at NTNU, measures which can streamline the review process are valuable. With a law change which requires at least two senior reviewers per exam submission taking effect from 1. August 2022, this need for streamlining will be even more pressing.

The goal is to present a design idea for a review tool, for student programming submissions, building upon the tools which are already developed for an educational use case by adopting techniques from how code inspection is done in the professional programming industry.

A thorough examination of our use case, the existing educational assessment solutions, and industrial review tools is conducted. Thereafter, comparisons between each use case are made to identify relevant features that are transferable to the proposed tool. These findings inform thesis' final contribution, a design proposal for the tool.

The contribution consists of a concrete sprint-based plan for how the desired review tool should be developed to enable an early release of a viable product and maximize the usability gained from each sprint cycle. In addition contributes the thesis by filling a gap in existing research in the field, as the background research showed that this use of knowledge transfer from industry to education, was not covered by existing research.

Sammendrag

Vurderingsressursene på NTNU er begrenset, tiltak som kan bidra til å effektivisere tiden brukt på vurdering er derfor verdifulle. 1. august 2022 trer en ny lov i kraft som krever at minst to sensorer skal samarbeide om vurderingen av hver eksamensbesvarelse. Denne lovendringen fører til en drastisk økning i behovet for effektiviseringen av vurderingsprosessene.

Hensikten med oppgaven er å komme opp med en design-idé til et vurderingsverktøy for studentkode. Dette verktøyet skal bygge på eksisterende vurderingsverktøy utviklet for bruk i utdanningssammenheng, samt låne teknikker fra verktøy for kodegjennomgang slik det er gjort i den profesjonelle programvareindustrien.

Det ble gjennomført en grundig undersøkelse av den tilsiktede bruken av et slikt verktøy, eksisterende vurderingsverktøy for utdanningsbruk og industrielle verktøy for kodegjennomgang. Deretter ble en sammenligning av karakteristikk ved bruks-situasjonene og motivasjonene bak bruken av de to kategoriene av verktøy gjennomført for å identifisere funksjonalitet hos de eksisterende verktøyene som vil være nyttig å overføre til et verktøy designet for vårt bruksområde. Når vi til slutt fremmer et design-forslag for et vurderingsverktøy er det er informert forslag med tyngde på bakgrunn av disse analysene.

Bidraget fra oppgaven består av en konkret ”sprint”-basert plan, hvis mål er å få på plass et brukbart vurderingsverktøy tidlig, deretter vil verdien inkrementelt øke for hver sprint-syklus. I tillegg bidrar oppgaven ved å fylle forskningsgapet på området, siden kartlegging av eksisterende kunnskap på området viste at denne konkrete bruken av kunnskapsoverføring fra industri til utdanningsfeltet ikke var dekket av eksisterende forskning.

Chapter 1

Introduction

Code reviews are an important part of system development. Code reviews are used in the process of adding and updating source files in a software project. When other developers inspect an author's code, errors that the author had missed can be spotted, through which the reviewing developers can help the code author learn and improve their knowledge. Since code inspection takes time and resources that could otherwise have been used for other important daily tasks, the professional industry has developed measures to make this process more time efficient.

It can be relevant to let the code inspection processes used in the professional industry inspire the way the review process in education is conducted, whether for feedback or for assessment. Since the professional industry is interested in streamlining a process that takes time from the developers daily tasks, it is expected that through many years of continuous review of the used tools, the industry has come up with tools of significant value and can therefore be beneficial for other fields to borrow inspiration from.

Based on the assumption that industrial code reviews have undergone years of development and streamlining, this thesis proposes that taking inspiration from industrial code reviews may also help to streamline the feedback and review process in educational programs for programming. The motivation for this master thesis is based on a wish from the lecturers at NTNU for such a system, that can help to streamline the feedback process of student submissions for some of the bachelor programming courses at NTNU.

1.1 Thesis Question

Given the potential value that code reviews tools developed for the industry could have for the educational fields, while bearing in mind that there are differences in the use-cases in the industry and in education, thesis will aim to answer the following question:

"How can industrial code review-tools inspire the development of a tool for streamlining the review process of student work at NTNU?"

Approach of the thesis

Our hypothesis is that the IT-industry has made well-considered choices when developing their tools used for reviewing of source code. By borrowing some knowledge from these tools, we can avoid designing a tool from scratch, instead building on existing knowledge. The goal of the thesis is hence to propose the development of an assessment tool for use in an educational setting at NTNU. The process for reaching this goal consists of multiple phases:

The first step is to understand the educational use case. This step consists of an examination of the experiences that the teaching staff has with the current solution for feedback and review, uncovering both the advantages and challenges associated with this solution. Following the analysis of the current feedback and review solution, the requirements of the target courses that this tool is being developed for will be examined.

The second step is to investigate the use case of code review in the industry, subsequently comparing and contrasting this with our educational use case. Since the educational use case may or may not share some characteristics with the professional use case, understanding similarities and differences is important to guide adaptation and customization of the knowledge gained from the industry code reviews to fit our use case.

The third step is to examine the various features that are currently in use in code review tools, both in the industry and in the educational use case.

The fourth step is to identify features being currently used in code review tools in the industry that can be useful to adopt in our educational use case.

The fifth step is to put forth a design proposal for the review tool.

Summarizing the above phases, thorough research will first be conducted to gather information from a range of relevant sources in order to answer the following questions:

- What is our use case, the actual problem we want to solve and current solutions for assessment at NTNU which we aim to improve upon?
- What other tools do already exist which are aimed at an educational use case?
- What are the similarities and differences between the use cases of code reviews in an industrial vs educational setting?
- What features are seen as valuable in an industrial code review tool?
- Are some of these features from the industrial code review tools also valuable to implement for the educational use case?

Subsequently, this research will be used to inform a design proposal for a assessment system.

1.2 Target Courses

The bachelor programming courses that we have in mind as targets for this reviewing tool are rather different. The structure of the courses and the expected submissions' form should play a role for the design decisions for the system. The target courses range from simpler courses where the submissions are created by a single author and consists of a single source code file, to courses with submissions that are more similar to the software projects' form used in the professional industry, which involve a team of developers and a more complex code base.

1.3 Definition of code review

In this thesis, code review (CR), is defined as the practice of a set of developers assessing an author's code. There are several motivations for conducting CRs. These motivations include correcting defects and knowledge transfer. CRs are proven to

be an effective tool for correcting defects (Silverthorne, 2021), while also being applicable for a wide range of developer team structures. CRs are for these reasons used as a tool for quality control in both commercial and open-source projects of all sizes.

Chapter 2

Background

2.1 The courses

To be able to create a useful and relevant tool that can streamline the review process, it is important to understand the full range of the various forms that the submissions for which the tool is intended to assess can take. As a guideline, I have analysed some of the courses where the use of a CR tool could potentially be relevant. This analysis is based on five courses currently offered at NTNU, and was conducted by inspecting the exercise descriptions for each of the courses and the exam for each of the courses with an exam.

2.1.1 Introduction Course to Programming

The first programming course at NTNU is Introduction Course to Programming (ITGK), where Python is used the programming environment (NTNU, 2021d), which covers the fundamentals of programming upon which future courses build. In this course, the submissions have a rather predictable form. The requirements for the student submissions are a series of functions to be implemented, and the input and output of the functions in the submission is controllable by the teaching staff. The code is rather simple and the submission usually consist of only one source code file.

2.1.2 Object-Oriented Programming course

The next two courses in line are two variations of courses in object-oriented programming.

- ‘TDT4100 - Object-Oriented Programming’ using Java
- ‘TDT4102 - Procedural and Object-Oriented Programming’ using C++

. These two course carries the same characteristics as ITGK in the way that the the input and expected output is controlled by the people that review the submission. The source code can often be more complex than in ITGK, and several files will usually be dependent of each other. The Object-oriented programming (OOP) course using Java has a smaller, more open ended project as its final exercise, this project has the characteristics of the submissions of the courses in the next paragraph, although being much smaller and less complex (H. Trætteberg, personal communication, 28. October 2021).

2.1.3 Software and Web Development Courses

The last two courses in our scope are software development (TDT4140 using Java) and web development (IT2810 using React.js). Compared to the other courses above, where assessments involve solving predefined problems where the function signatures are provided, the projects in these courses are more open-ended. For example, students may be asked to create an application for use in a particular industry, and have the freedom to define the scope of their own project. As a result, the submissions for these assessments are less standardized, and the source code that students produce differ greatly. The assessment of these submissions are therefore more complex. A description of the kind of system is supplied by the staff, but it is not possible to for the staff to write unit tests that can work for every project. Usually the student teams are suppose to write their own unit tests. The projects in these courses are done by teams of multiple students.

Comparing and contrasting the assessments used in the courses above, there are differences in the nature of the submissions, and subsequently the complexity involved in the assessment processes. Since the Introduction to Programming and Object-Oriented Programming courses consists of a defined set of method specifications, the behaviour of each individual function implemented in a student’s can be

checked using unit tests. On the other hand, the requirements for the submissions in software and web development are only in regards to what technology to be used and the type of industry the application is belonging to. The use of unit tests to streamline the review process are therefore not applicable for these two courses.

While the analysis is based on the aforementioned five courses currently offered in NTNU, it is worth considering that sometimes new programming courses are offered at NTNU, and some of the old ones are changed. Bearing this in mind, building a tool that can be helpful for a wide range of submissions could be beneficial in the future.

2.2 Formative and Summative assessments

In the educational system, faculty and teaching staff conduct reviews for different kinds of student work, with different motivations. This places a wide range of different requirements for the various features in our proposed system. For example, for most assessments and review of work throughout the semester (apart from review of the final exam), the main objective is to facilitate the identification of knowledge gaps and eventually knowledge transfer, in order to guide the student to achieve the learning goals of the course. On the other hand, the reviewing of the final exam of a course focuses on measuring the student's knowledge and attainment of learning goals at the end of the course. The former type of assessment is referred to as formative assessments, while the latter type is termed summative assessments. The following sections further explore the differences between formative and summative assessments.

2.2.1 Formative assessment (assessment for learning)

The formative assessment is conducted during the learning period. The goal of the FA is to facilitate learning (Helle and Burner, 2021). Pointing out what mistakes were done during the assessment situation are not the end in itself, but instead a means to helping the student see the path forward. The feedback from the formative assessment needs to tell the student what remains to fill the gap between the current level of knowledge shown, and the required level. In addition the feedback must contain instructions on how to reach the required level.

2.2.2 Summative assessment (assessment for control)

Summative assessment is conducted at the end of a learning period. The goal of the summative assessment is to control if the student possesses the necessary knowledge to move on from a subject.

Taras (2005) states that an assessment can be purely summative, if it focuses solely on assessing performance and identifying mistakes, without processes to bridge knowledge gaps. On the other hand, an assessment can not be purely formative, as a knowledge mapping of the student work has to be done before the instructor can show the student the path forward. This summative assessment as part of an formative assessment can either be available for the student, or it can be undisclosed to the student.

2.2.3 Assessment types in our use case

Throughout the course semester, students undergo both summative and formative assessments.

2.2.3.1 Formative

At NTNU most courses conduct mandatory weekly exercises, which are usually not part of the final grading of the student. These weekly exercises are therefore a formative assessment, whose goal is to help students improve their knowledge, correct misconceptions and show them the path forward. As CRs bring a lot of pedagogical value, and one of the main motivations for doing CRs is knowledge transfer, it makes sense to use a tool inspired by professional CRs for a process like formative assessment, where the goal is to facilitate learning.

2.2.3.2 Summative

In most courses at NTNU, the final exam is the only summative assessment upon which the final course grade is based. Some courses do also conduct a mid-term exam in addition to this final exam, which bears the same form as the final exam, and can hence be grouped together with the final exam as the group of exams. Since the goal of such assessments is solely to measure and assess the student's knowledge

(Helle, 2020), these exams are summative assessments. Accordingly, the knowledge transfer is not the focus for this group of exam submissions. Therefore, for such settings, the aspects of the CR-tools that facilitate knowledge transfer are not in the central focus. However, in addition to knowledge transfer processes, CR-tools also have functionalities which streamline the review process itself, making it useful to take inspiration from CR tools for this group of submissions.

2.2.4 The reviewer in formative and summative assessments

According to educational law, the person conducting the review of the exam must meet certain qualification requirements. This means that while the weekly exercises can be evaluated by the many teaching assistants (TAs), summative assessments used in the grading of the course can only be reviewed by a limited number of qualified people in the course staff. The proposed tool will therefore be intended for summative assessments for use of course staff, since it is for these kinds of submissions that there are a limited amount of reviewing resources and therefore the need for streamlining is in highest demand.

In some cases, our proposed tool would also be necessary to support the review of formative assessments. Formative assessments at NTNU is usually in the form of weekly exercises or longer running projects. Some of the courses use these weekly exercises or projects as part of the final grading of the student for a course. Since the law states that only the qualified staff can conduct reviews of assessments used in grading, the reviewing conducted by the TAs throughout the course cannot be directly used as part of the grading process, resulting in similar resource constraints. However, the TAs' reviews can on the other hand be utilized as data in the grading of the submissions. By basing their review on a checklist of requirements for submission, there is little need and room for the TAs to exercise subjective judgment in grading a submission. In this way, TAs' reviews can be used by teaching staff as data for the final course grading. This reviewing strategy saves scarce senior staff resources while remaining inside the law.

2.2.4.1 Educational law change

From the 1st August 2022, a new law requiring two sensors for each exam will take effect (Kunnskapsdepartementet, 2005). This gives rise to a need for a platform for the sensors to collaborate on the reviewing of the exams. As the industrial CRs are

usually conducted by multiple reviewers asynchronously, this is yet again a use case where it is relevant and useful to take inspiration from their practice.

2.3 Current system for reviewing

The reviewing process for programming submissions can vary between courses, and between the reviewers of the same course.

Based on e-mail conversations with the lecturers and course coordinators in the relevant subjects, how the reviewing process differs between the courses was mapped out.

2.3.1 Introduction Course to Programming

In the introduction course to programming (ITGK) using Python as the programming environment, the formative assessment of the weekly programming exercises are conducted through a process where the student demonstrates the code to a TA. This process bears resemblance to the CR technique called the *over-the-shoulder* process (Cohen, 2006), as the student is explaining the workings of their program while the TA is giving feedback. The student and TA then collaborate to find a solution if it is needed.

Regarding the summative assessments in the course, there is only one, the final exam. The exam consists of a series of multiple choice questions and as the main part - a series of functions to be implemented from scratch. The multiple choice questions are targeting both theory knowledge and code understanding (Appendix E). As the programming tasks are rather not so complex, they are evaluated by just reading the source code and the reviewer visualises the program flow in its head, with no use of support tools. The multiple choice questions are automatically evaluated by the exam system (H. Trættemberg, personal communication, 5. Dec 2021).

2.3.2 Object-Oriented Programming

In the object-oriented programming courses using C++ or Java as the programming environment, the formative assessment of the weekly programming exercises are conducted in the same way as in ITGK, being that the exercises are demonstrated

to a TA. The exam of the two courses consists mainly of a series of functions to be implemented, but also some theory questions and code understanding (Appendix F & Appendix G).

For the summative assessment, the students upload a zip compressed version of their exam submission. The reviewer later downloads the zipped submission, and the code is run through a series of unit tests. Subsequently, the submission then achieves a score for each test. The reviewing solution currently in use then generates a summarizing report in the form of a HTML-document (Appendix B). The HTML solution is a bit different for the course using C++ and the one using Java, where the former also displays the model answer (instructor’s intended implementation) in the HTML-report along the student’s solution, while the latter does not. The solution currently in use is explained more in detail in Section 2.8.2.

Feedback from the lecturer in the C++ course was that the system could also include a process where the scores from each of the unit tests are gathered, and a letter grade subsequently suggested based on these unit test scores (Appendix B).

2.3.3 Software Development

The assessment of a student in the software development course is solely based on a single big project, as the course does not have a final exam. The assessment is mainly based on the development team’s process, as agile software development is an important part of the curriculum. The technical skills are also part of the course content, “...*software process implementation, software evolution and maintenance, software reuse, ..., software quality, ..., software verification and testing, software architecture.*” (NTNU, 2021e) In the software development course, the submissions are usually examined by either cloning the students’ projects to the reviewer’s machine, or reviewing the code directly through GitLab’s project interface (H. Trætberg, personal communication). In both cases, current reviewing processes are not supported by external tools that could help to streamline the reviewing process. In addition, cloning repositories locally takes space and time. A process that allows reviewers to avoid doing so would be beneficial and increase the efficiency of the process.

Furthermore, since the comments from the reviewer are currently written in a separate file, there is no visual connection between the comment and the actual code line(s). This current system for reviewing can be improved by implementing a way

to visually connect comments to the actual code line(s), which makes it easier for the students to get learning outcome from the comments. By adopting some of the choices made in professional CR tools for commenting on code, the reviewing tool can be more intuitive and time saving for the reviewer than what is currently employed (e.g through drag-select or select a single code line through click.)

The potential ways of streamlining the review process of this course is by make it easier to navigate the project's file structure, between the involved source code files, as the projects submitted for this course is larger and more complex than the ones in the courses mentioned above.

2.3.4 Web Development

For the web development course (NTNU, 2021a), the exam usually does not consist of programming tasks, as the mandatory projects already cover the mapping of the students' programming knowledge and skills to a great extent. Instead, there is a greater focus on mapping the students' theoretical knowledge, familiarity with the relevant tools, and ability to reflect and think critically about the choices to be taken in the domain. The exam submissions therefore consist primarily of text answers and, in some cases, drawn figures.

In this course, the students do peer reviews of the multiple projects throughout the semester, as part of the mandatory course projects through the external service "*Eduflow*" (Eduflow, 2021). The students deploy their web application to the service "*Gitpod*" (Gitpod, 2021), which is also the platform where the reviewers will visit the application. The code will be inspected by either cloning the project repository hosted by GitLab or inside the repository view in GitLab. The feedback comments are then written in Eduflow. The current process is rather cumbersome, involving a total of three services and platforms. In addition, an important weakness of this solution is that the comments are not visually connected to the code lines targeted. Based on e-mail communication with the lecturer from the web development course, their need and wish is for a tool that supports the peer CR processes that are currently employed in the course (Appendix C). In particular, this tool should automatically assign peer reviewers and in an anonymous way. The senior staff for the course should have reading access, so that they are able to follow and monitor the review process. Since the students reviews are used as a data base for the final grading of the course, it is important that the authors have the opportunity to flag

and respond to reviewers' comments which they find to be unfair and/or wrong. In addition, the lecturer for the course writes that they are mostly interested in features that facilitates active student assessment forms (Appendix C). By active student assessment forms, we mean forms of assessment where the students are assessing each other (Burner et al., 2011).

The teaching staff have decided that peer review of other students' projects is an important part of the curriculum in the web development course, with the intention of allowing students to be exposed to and engaged in the CR process. However, following the change of law discussed in Section 2.2.4.1, in order to keep the peer review component, the staff is planning to change the grading of the course to pass or fail. By the course being pass or fail, the new law of two sensors will not be required by law, and it will be easier to continue with the practice of using students' peer reviews as data base for the final grading of the course.

The benefit of using our tool for the exam in this course is arguable, since the tasks in the final exam of the web development course have historically been more focused on the mapping of theoretical knowledge and knowledge of relevant tools, as well as making reflective choices in the domain, while our tool on the other hand is aimed at reviewing of source code.

For the peer review of the student projects on the other hand, the course can benefit from gathering the inspection of source code, launching of the web application and the reviewing interface in one application. The benefits this brings are that the comments will be visually connected to the code, and the senior staff will have the possibility to view all student reviews for a submission in one view.

There is therefore a huge potential for improvement in the current CR-process, which steals time from the other important daily tasks. In order to help the teaching staff to save time and other resources, which is particularly crucial since the senior staff resources are limited, we can take inspiration from a professional industry in streamlining this resource-consuming process.

2.4 Value of Code Review-technique exposure for students

While our main motivation for bringing the CR techniques into the educational domain is to streamline the review process, a side effect is that students are exposed to such CR-tools. There is value in such exposure, since these are techniques which the students will meet later - not only in later programming courses at NTNU but also while working in the industry.

Individuals learn from experience, and experiential learning is a powerful pedagogical tool (Bradford, 2019 and Kolb, 2014). Kolb's *Experiential Learning Theory* (2014) highlights a four-stage learning cycle, where learners go through a *concrete experience*, engage in *reflective observation* about the experience, formulate an *abstract conceptualisation* or draw lessons from the experience, and finally engage in *active experimentation* and applying these lessons (Kolb, 2014, p. 50-52). Early exposure to CR-tools in their academic journey provides an initial *concrete experience* for students in CR-tools, triggering subsequent processes of *reflective observation* and *abstract conceptualisation*. When students encounter CR in their future courses or in their professional careers, they can build upon the learning gleaned from their prior experience with CR and engage in *active experimentation*, kick-starting the experiential learning process. Even though the specific tools used in CR might differ, the objectives, functions, processes, and mindsets involved will be similar, and learning from such exposure will be transferable to future settings to a significant degree.

2.5 Motivation behind industrial CRs

It is meaningful to be aware of the motivations behind the design choices of the CR-tools used in the industry. On one hand, some of the features in industrial CR-tools are there because of motivations and objectives that are relevant in the industrial use case, but not in our educational use case. On the other hand, some motivations and objectives are relevant for both the industrial use case and the educational use case. To understand which features implemented in industrial use cases are relevant for us in the educational use case, I first analyse the specific motivations and objectives behind the use of CRs in the industry in this section, subsequently comparing and contrasting objectives between both use cases in the following sections, before finally



Figure 2.1: Developers' motivations for CR (Bird and Bacchelli, 2013, Figure 3)

identifying the features that are most relevant to fulfilling these objectives.

2.5.1 Motivation in Microsoft behind doing CRs

Based on a report on motivation behind CRs (Bird and Bacchelli, 2013), we can see that finding defects and improving software quality are the primary motivations for conducting CRs, both for the managers and the developers. The report also brings up secondary motivations for conducting CRs, including:

- Knowledge transfer
- Platform for suggesting alternative solutions
- Platform for enforcing code conventions
- Making the developers less protective about their code
- Team awareness and transparency
- Shared ownership of the code

2.5.1.1 Finding defects

Finding and correcting defects is an important reason for doing CRs. To be exact, 44% of the developers and managers interviewed ranked finding defects as their number one motivation for implementing CRs (Bird and Bacchelli, 2013, p. 5).

Defects in this context includes bugs and mistakes in the design. According to a developer interviewed, exactly what tool is used to support the code inspection doesn't matter so much for their team, as long as it can help the developer identify defects. CRs are in some occasions even claimed to be a cheaper way of identifying bugs, compared to regular testing (Bird and Bacchelli, 2013, p. 5).

2.5.1.2 Code improvement

Based on Bird and Bacchelli's (2013) report, the effect of CRs on code improvement is also an important benefit for many, with 39% of the interviewed developers ranking code improvement as their main motivation. This proportion is slightly lower amongst the managers, but still significant, at 31%. In this context, reviewers highlight how the code can be improved, covering issues such as readability, consistency in design and dead code (code that should have been removed since it is not ran). The use of CR also improves code in an indirect way. Knowing that one's code will be subsequently be inspected by one's colleagues can provide motivation for developments to focus more on writing high quality code and explaining it to others, adding a healthy pressure to produce quality, which is of value in itself (Bird and Bacchelli, 2013, Chapter IV.B).

2.5.1.3 Platform for suggesting alternative solutions

While suggesting alternative solutions is very linked to both code improvement and knowledge transfer. It is one of the motivations behind CRs which are not equally emphasized by the developers and the managers. 17% of the developers saw CR being a platform for suggesting alternative solutions as their main motivation, while none of the managers had this as an motivation. Alternative solutions is in this context solutions that leads to a better implementation (Bird and Bacchelli, 2013, p. 5).

2.5.1.4 Knowledge transfer

Using CRs as a platform for sharing knowledge across the team was also a motivation for all the interviewed (except one interviewee). CRs are known for being looked at as mainly a way of teaching new developers code writing and the conventions used in the team (Bird and Bacchelli, 2013, p. 5), and not so much as a tool for

teaching the more experienced. However, based on numbers from the report, we can see that most developers appreciate the value that CRs have for their learning. As an interviewee explains, "If you do a CR and did not learn anything about the area and you still do not know anything about the area, then that was not as good CR as it could have been." (Bird and Bacchelli, 2013, p. 5).

2.5.1.5 Team Awareness and Transparency

Fostering team awareness and transparency are closely linked to knowledge transfer, but differs in an important way. While knowledge transfer is mainly a transaction between the author and a reviewer, the objective of creating team awareness and transparency involves creating access to the code for all, and notifying everyone about any changes being made to the code. In this case, everyone gets the opportunity to learn techniques that can improve the entire code base, not just the author. An example of this is cited in the report, where a developer was notified of a CR that had been conducted on another author's code. When reading this CR, the developer was exposed to a smart solution that the reviewed had proposed, and subsequently applied the solution in his own code (Bird and Bacchelli, 2013, p. 6). Team awareness and transparency can also prevent people from making changes to the code which could potentially break the code, for example by tracking changes made by the team or notifying team members when a change has been made.

2.5.1.6 Shared Code Ownership

The reason why shared code ownership is a motivation is to make sure that multiple individuals are having knowledge of each part of the code base. CRs allows developers to get to know parts of the code base that they usually don't interact with, which helps the company to prepare "back-up" developers in addition to the authors, who have knowledge of the code (Bird and Bacchelli, 2013, p. 6). In this way, CRs can help avoid situations where only one person knows a part of the system, so that in the event that this expert leaves the company, they are not left with zero knowledge about that part of the system.

A side effect of creating shared code ownership highlighted by Bird and Bacchelli (2013) is that developers tend to be less protective about their code and more open to critique if there is an understanding that others will be reading and reviewing the code (Bird and Bacchelli, 2013, p. 6). A total of 27% of the interviewed had shared

code ownership as one of their top three motivations for CRs, a much lower figure than for the other motivations mentioned above.

2.5.2 Actual use of Code Reviews in the industry

As part of the report, the researchers analysed close to 600 comments from actual CRs conducted by developers in the department in review. The analysis revealed that even though defect identification was the biggest motivation behind CRs for both the developers and managers, it was in fact not what CRs was mainly used for. Just 78 of the comments were concerning code defects, while comments on code improvement was the most frequent category with over twice as many comments at 165. Even though the authors argue for the knowledge transfer and other outcomes of CRs as being abstract and social, they found in total 12 comments they classified as knowledge transfer. The frequency of code improvement and knowledge transfer comments reveal that CRs do in fact have an important pedagogical function in the professional industry. While developers and managers expect to use these tools mainly for identifying defects, these tools are being more frequently used for teaching better code practice and improving readability (Bird and Bacchelli, 2013, p. 7).

Bird and Bacchelli (2013) also highlight a potential area for improvement for CRs, where many of the comments were concerning breaking of code conventions, typos and identifying dead code. All of these and other problems targeted in comments are easily automatable by already existing tools for static code analysis. By using tools like this, the reviewer saves time that can rather be used on focusing on problems that are deeper and more difficult for the static code analysis tools to identify (Bird and Bacchelli, 2013, p. 9).

The authors also assert that developers will benefit from having richer ways of communicating with their peers. They argue that asynchronous communication through comments in a CR-tool is insufficient, and propose discussing the solutions face-to-face or through other synchronous ways of communication (Bird and Bacchelli, 2013, p. 9).

Finally, the authors highlight that when the reviewer is familiar with the code and the context of the code from before, both the efficiency of the review and the value of the comments are higher than if they did not have prior knowledge (Bird and Bacchelli, 2013, p. 9). This points to importance of keeping the team updated on each others work. This is relevant for our use case as well, as it implies that the

reviewers will do a better review for the tasks they know better.

2.6 Industrial Code Reviews and Our Use Case

Having explored the motivations and objectives behind CRs in the industry, the next step is to compare and contrast the use cases of CR-tools in the industry and the educational use case. Since our use case is quite different from the use of CR-tools in the industry, it is important to first analyse the similarities and differences in both the motivations and source code-in-review between our use case and those in the industry. This will help us to identify which features of the CR-tools in the industry are relevant for our educational use case. Therefore, the following section will compare the two use cases.

2.6.1 Motivations and Objectives

Summarising insights from the software development industry, there are two main motivations for conducting CRs. The most obvious reason is to assure code quality and avoid bugs in published software. By having the code checked by another set of eyes, mistakes that the original author missed may be revealed. Furthermore, project requirements may be interpreted differently by each developer. By having other members of the team go through your work, any misinterpretations can easily be caught, ensuring that the code developed is aligned with the team's unified understanding of the project. Furthermore, knowing that one's colleagues are going to review one's code can motivate the developer to ensure that the code is well-designed and that all of the tests are running (Bird and Bacchelli, 2013, Chapter IV.B). This will benefit the team in the long run, since well-designed code will also encounter fewer bugs (Radigan, 2021).

One key objective of the desired CR-tool in the educational use case is for the grading of examination submissions in summative assessments. Similar to the objectives in the industry, this involves the identifying bugs and mistakes in the code to identify knowledge gaps and assess whether the student has fulfilled the learning objectives. In addition, it also involves ensuring that the student had understood the assignment right, and that the code being developed fulfills the requirements of the task. Features in the industry's review tools that facilitate the identification of bugs and mistakes, as well as the assessment of how appropriate the interpretation of the task

was, are relevant for our proposed tool.

The second reason for conducting CRs in the industry is knowledge sharing. Since a software development team consists of team members with varying levels of experience and from different fields of expertise, CRs will in many cases be beneficial for developing programming and other related skills by facilitating learning from one another. Furthermore, the tech sector, and the IT sector in particular is a field that is in constant development, and CRs are a great way to keep up with the latest techniques. If an author has implemented a solution in an outdated way, and the reviewer knows of a smarter solution which utilises the newest state-of-the-art one-liner, the reviewer is able to share this knowledge with the author, helping him to stay up to date. The same goes vice versa, where the reviewer can also pick up smart practices from reviewing code written by others. This will likely lead to the team writing more efficient and more easily readable code - better code. In addition, sharing of knowledge is not just about teaching new smart solutions for problems, but also about incorporating a common coding style for the entire team. Just as a written report should use the same "voice" throughout the entire report, a programmed project should also use the same voice for all parts of the project. This makes it easier to understand each others code in the team, since everyone uses the same style.

On the other hand, another key objective of the desired CR-tool in the educational use case is to facilitate learning through the formative assessments. While this involves highlighting to students the mistakes made in the code, it also involves explaining why it was a mistake and how these mistakes can be rectified. In addition to identifying and correcting bugs, students will benefit from being shown how their code can be improved or written in a better and more efficient way. These objectives bear similarities to those in the industry use case. Features in the industry review tools that enable reviewers to highlight mistakes, explain the mistake and communicate a right or better solution are relevant for our proposed tool.

2.6.2 Source-code in review

In the industry, CRs are usually done in connection to the event of updating or adding to an existing code base (pull request) with new code. In this case, the reviewers are usually already familiar with the code base, and it is therefore easier for them to see how the new code fits into the existing code. In addition, changes to

the code are also performed incrementally. Furthermore, since reviews target merely the latest changes to the code, the entire program is not inspected at each review.

In our use case on the other hand, students sometimes receive a code skeleton, but the rest of the code is added by the student. Hence, a large fraction of the code is new for the reviewer. For the subjects which have larger project and relatively few submissions, this also means that a significant amount of code is written and has to be reviewed during the same review. The way the source code (and belonging documentation files) evolves in the educational context, from the time of skeleton code being handed out to the time of review is different than in the industrial use case. For most deliveries in the courses in scope is the entire student contribution done in this one interval, with no incremental changes to the files to review. For some submissions are the students asked to re-deliver a revised version of the original submission with corrections based on feedback from an "en-route" assessment of the initial version. These re-delivered submissions does hence a common characteristic with the industry code-in-review, which is expected to be reflected in the need for features which support this way of code evolving between two reviews in addition to the more regular way of the code evolving entirely between hand out and review. This rather important difference leads to a different need for features.

Since the CRs in the industry usually is concerned with just the parts of the code that are altered, the need is for a (commenting) feature which facilitates thorough/detailed communication with your co-developers about just a few small changes. In our use case, the need is for a commenting feature that makes it easy to comment on many changes / blocks of code.

2.7 Features from industry tools

2.7.1 GitHub

GitHub is the largest source code host in the world, and is hence an essential part of the software development cycle for many development teams. GitHub has their own implementation of a CR supporting tool, this functionality is usually in use in connection to pull requests - updating the old source code version with the new changes (GitHub, 2021). GitHub is aimed at development teams of all sizes. The user has the choice to either show the code comparison / diff in a split view or a unified view. In the split view, as shown in Figure 2.2, the modified or removed

```

139 - <InputField {...register("name", {required: true})} name="name" placeholder="Navn Navnesen" autoFocus/>
140 - <InputField {...register("email", {required: true})} name="email" placeholder="navn@domene.no" />
141 - <InputField {...register("phone", {required: true})} name="phone" placeholder="(+)XX XX XXX" />
142 - <InputField {...register("areacode", {required: true})} name="areacode" placeholder="0000" />
143 - <Commentbox {...register('comment')} name="comment"/>
144 </InputsContainer>
145 </FieldsContainer>
146 <SubmitButton type="submit">Send inn!</SubmitButton>
147 </FieldSet>
148 </form>
149 </FormContainer>
150 {showErrors ? (<ErrorMessage error_bools={errors} showErrors={showErrors} doClose={() => clearSubmit()}>): null}

169 + <InputField {...register("name", {required: true})} className="required" name="name" placeholder="Navn Navnesen" autoFocus/>
170 + <InputField {...register("email", {required: true})} className="required" name="email" placeholder="navn@domene.no" />
171 + <InputField {...register("phone", {required: true})} className="required" name="phone" placeholder="(+)XX XX XXX" />
172 + <InputField {...register("areacode", {required: true})} className="required" name="areacode" placeholder="0000" />
173 + <Commentbox {...register('comment')} name="comment"/>
174 </InputsContainer>
175 </FieldsContainer>
176 <SubmitButton type="submit">Send inn!</SubmitButton>
177 </FieldSet>
178 </form>
179 </FormContainer>
180 {showErrors ? (<ErrorMessage errors={errors} showErrors={showErrors} doClose={() => clearSubmit()}>): null}

```

Figure 2.2: GitHub: Source code changes presented in a split view

```

139 - <InputField {...register("name", {required: true})} name="name" placeholder="Navn Navnesen" autoFocus/>
140 - <InputField {...register("email", {required: true})} name="email" placeholder="navn@domene.no" />
141 + <InputField {...register("phone", {required: true})} name="phone" placeholder="(+)XX XX XXX" />
142 - <InputField {...register("areacode", {required: true})} name="areacode" placeholder="0000" />
143 - <Commentbox {...register('comment')} name="comment"/>
169 + <InputField {...register("name", {required: true})} className="required" name="name" placeholder="Navn Navnesen" autoFocus/>
170 + <InputField {...register("email", {required: true})} className="required" name="email" placeholder="navn@domene.no" />
171 + <InputField {...register("phone", {required: true})} className="required" name="phone" placeholder="(+)XX XX XXX" />
172 + <InputField {...register("areacode", {required: true})} className="required" name="areacode" placeholder="0000" />
173 + <Commentbox {...register('comment')} name="comment"/>
144 </InputsContainer>
145 </FieldsContainer>
146 <SubmitButton type="submit">Send inn!</SubmitButton>
147 </FieldSet>
148 </form>
149 </FormContainer>
150 {showErrors ? (<ErrorMessage error_bools={errors} showErrors={showErrors} doClose={() => clearSubmit()}>): null}
180 + {showErrors ? (<ErrorMessage errors={errors} showErrors={showErrors} doClose={() => clearSubmit()}>): null}

```

Figure 2.3: GitHub: Source code changes presented in a unified view

code lines is highlighted in red in the file representing the old version. Additions or modifications from the old version has green highlighting in the file view representing the new version.

GitHub does also support a unified diff view (Figure 2.3), where the changes are shown in one file view, with both the red (old) and green (new) stacked on top of each other. In this way, it is easy to see what code replaced the old code line. If an GitHub App is added to the CI-pipeline as a "Check", the annotation from these can be toggled so they show up inside the file view during a pull request (Figure 2.4). Among these GitHub Apps are linting tools and other types of static code analysis tools.

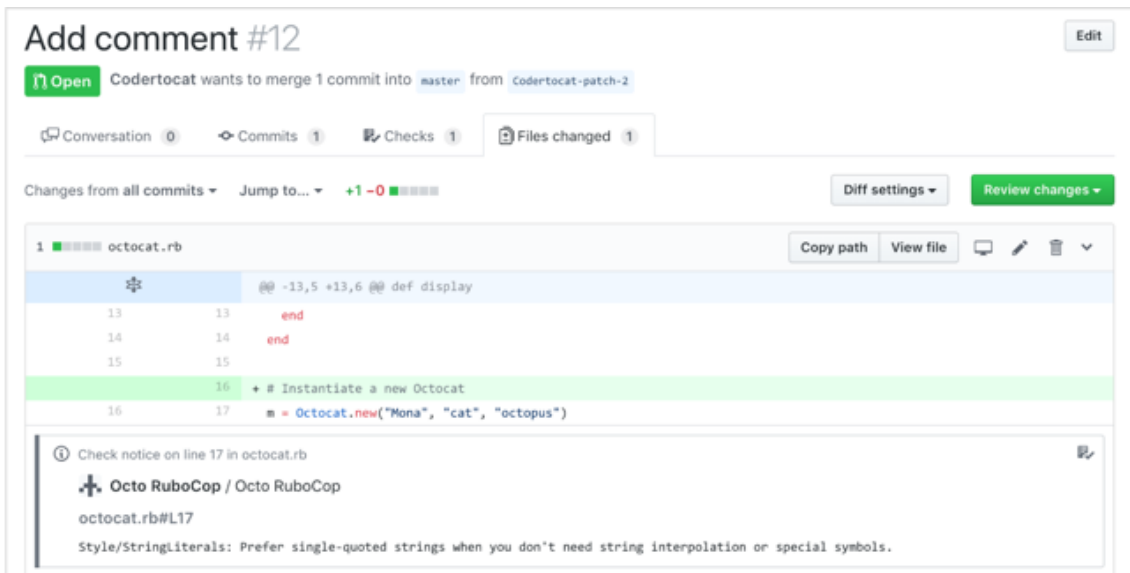


Figure 2.4: GitHub: Code quality annotations in review view (Github, 2021)

2.7.2 GitLab

GitLab is a source code hosting tool such as GitHub. GitLab is as of November 2021 the source code hosting tool used at NTNU as a default for most of the courses. According to GitLab, from their own page (GitLab, 2021), the most important features in an CR-tool depends on the use case. The motivation for using a CR-tool is to ensure code-quality in the production code. In addition, another important motivation for many is the learning aspect, where examining each others code in the team makes a good opportunity to share knowledge across the team.

A good CR-tool, according to GitLab, is a tool that emphasises collaboration, not only shipping quality code. The tool should implement a comment feature that sparks discussion. If the team decides that the collaboration is particularly important, then implementing possibilities for taking notes and commenting on changes can be used for fruitful discussions later. Since a team rarely has time to sit down and discuss changes at the same time, it is important that the team members can document their ideas for others to read at a later time. To get a team to use your CR-tool, it is important that the tool can fit in in the ecosystem of tools used by developers today. This means that integration of the most used tools for version control and inspecting merged code is important. A tool should at least work seamlessly with Git, since Git is by far the most used version control system (VCS). A CR-tool that wants to bring value should have the ability to collect and report key metrics about the code. One should be able to see changes done from last iteration

(e.g though diff-file). A more advanced tool should also report on code violations (linting mistakes, bad coding practices, etc.) If a team only has to use one tool for conversations about the code, then this will save the team a lot of time.

GitLab's philosophy is that a developer will save time by reducing the number of tool used, based on this rule, GitLab has made an extension for Visual Studio Code which enables the developer to conduct the CR directly in their code editor (O'Leary, 2021). Since GitLab is responsible for the hosting of the source code and continuous integration / -development (CI/CD), the CR part of GitLab does benefit from everything happening in one system. If the team has configured code quality requirements in the CI-pipeline, these will show up as indicators (Figure 2.6) next to the code quality rule violating piece of code in the diff view, and can hence to easy addressed by the reviewer and corrected. GitLab has multiple ways of showing the diff files dependent on the user's preference. Just as in GitHub, the user can decide if it wants to show the new and old version side by side in two file views (split) or as a unified view. Deleted code and added code is highlighted respectively in red and green. GitLab does also have a setting for showing one file at the time. The default setting is that all the files that are affected by changes is presented in a vertical list view, but by changing this setting to showing one file at a time, it might be easier to assure that no changes is left unnoticed. In the same setting view, the user does also have the opportunity to display the file structure of the project in a flatter view, such that the directory structure depth is shown as a string representation instead of a tree view with a lot of indentations. The flattened file structure view is chosen in Figure 2.5 GitLab has a simple check box (upper right corner in Figure 2.5) at the top of each file that makes it easy for the reviewer to mark for themselves what files they have inspected and which they have not.

2.7.3 Collaborator

Collaborator is a dedicated CR-tool made by SmartBear software. They claim to have Adobe, Cisco, Citi, Oracle in their user base (Smartbear Software, 2021a). Collaborator offers integration with IDEs including Visual Studio Code and Eclipse, code hosting services such as GitHub, GitLab and Bitbucket, and a range of tools for making documentation and diagrams including Microsoft Word, PowerPoint, Excel and MathWorks' Simulink and some bug trackers (Smartbear Software, 2021b). In addition to the traditional red and green highlighting of source code changes, Collaborator also implements the use yellow highlighter for code lines which are

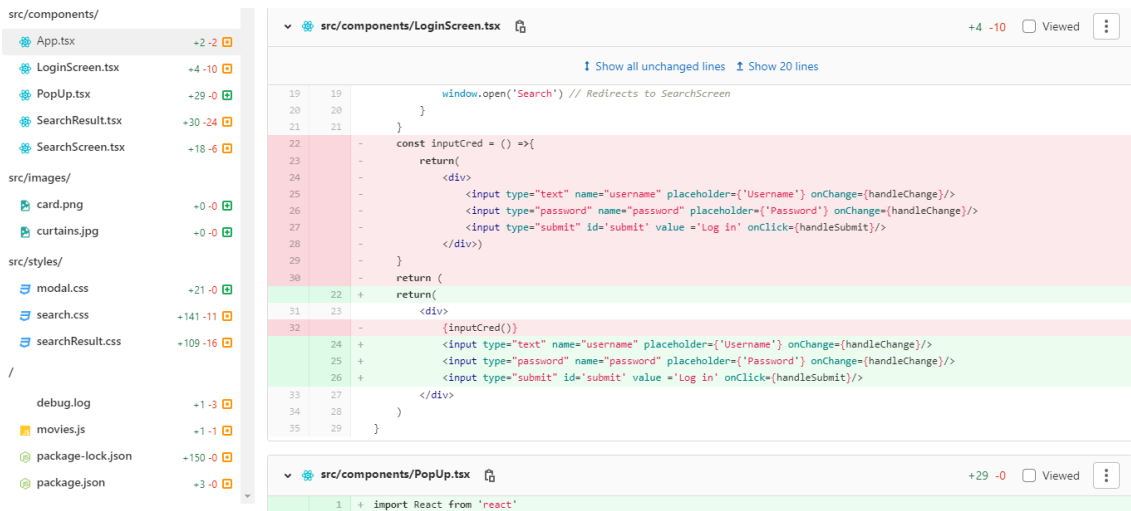


Figure 2.5: GitLab: CR interface



Figure 2.6: GitLab: Code quality indicators inside the review interface

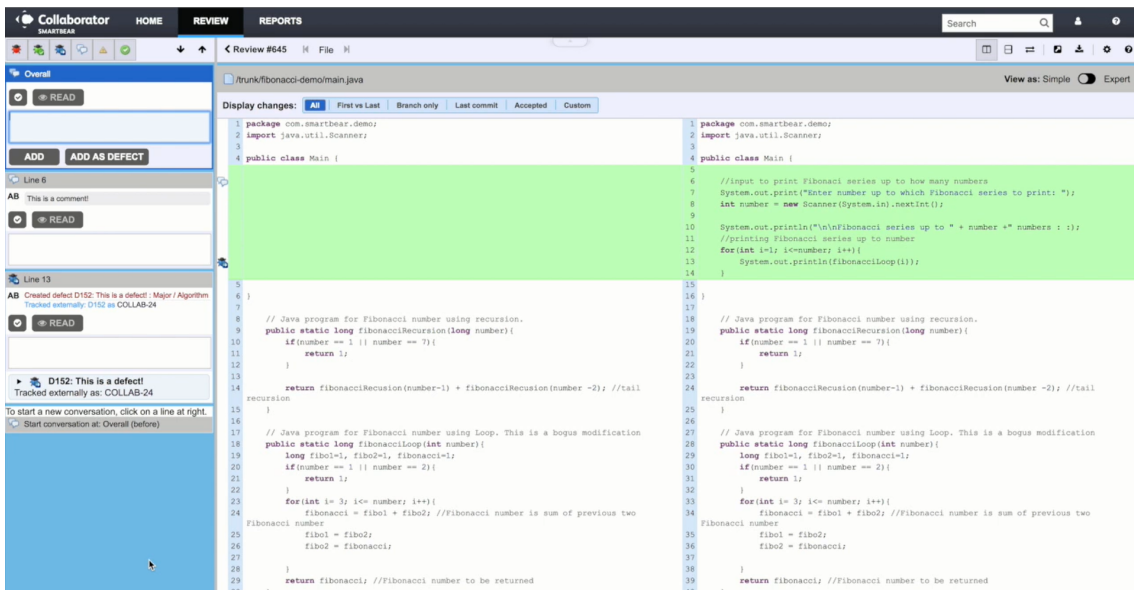


Figure 2.7: Collaborator: CR interface

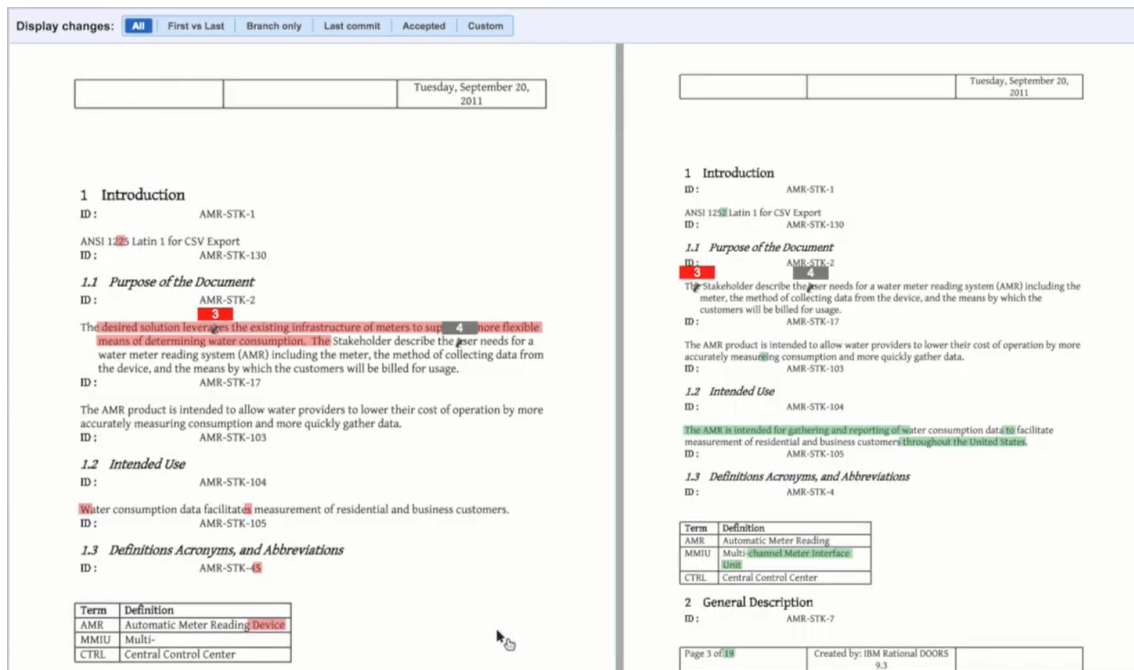


Figure 2.8: Collaborator: Review interface for PDF-documentation file

modified. In Collaborator, the reviewer can leave two types of feedback comment - discussion and defect. These comments are added in a pane at the left side of the file-in-review. In Figure 2.7, both comment categories are shown, discussion comments with a speech bubble-icon and defect comment with a lady bug icon. In most of the other professional tools the comments are added inline. The presentation of the comments in Collaborator does hence have closer resemblance to the presentation of comments in a educational tool we explore later, CodePost (see Section 2.8.1), than in the other professional industry tools.

The discussion comments are feedback that you as a reviewer want to leave, but you do not require the author to respond with a change to that comment. Defects on the other hand is comments that address issues that has to be resolved before the review can be approved. This differs from the way reviews are done in GitHub, where the entire review can be an issue that has to be responded with changes for the pull request can be merged. In Collaborator the granularity in the review is higher, as the reviewer can tell the author exactly what issues which has to be addressed before the source code is at an acceptable level.

As explained earlier, Collaborator supports as wide range of documentation tools. The tool is able to do file diff for files from Microsoft Word, PowerPoint, Excel and PDFs (Figure 2.8) (Smartbear Software, 2021c). The users can then address content in the documentation files by setting push pin, and write their comments in

```

package tds4145;
import java.sql.*;
public class Driver {
    static ResultSet result = null;
    public static void Write(String query){
        try {
        } catch (Exception e) {
        }
    }
    public static void PrintTable(String table) throws SQLException {
        ResultSet rs = null;
        try {
            rs = DriverManager.getConnection("jdbc:tds://localhost:1433/Adventureworks2008").createStatement().executeQuery(query);
            while(rs.next()) {
                System.out.println(rs.getString(1));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static ResultSet Read(String query) {
        try {
            Connection myConn = DriverManager.getConnection("jdbc:tds://localhost:1433/Adventureworks2008");
            Statement statement = myConn.createStatement();
            result = statement.executeQuery(query);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void PrintSet(ResultSet result) throws SQLException {
        int columnCount = result.getMetaData().getColumnCount();
    }
}

```

Figure 2.9: Bitbucket: Source code view

the left pane in the same way as for comments on code. For the people concerned with insight about the CR process, Collaborator does have metrics about how long time each reviewer has used on the CRs and a measure for the inspection rate (lines-of-code per hour).

2.7.4 Bitbucket

Bitbucket is Atlassian’s solution for source code hosting (Atlassian, 2021a), and serves in many ways the same purpose as GitHub and GitLab. Bitbucket is web based. CRs are conducted in relation to merging source code branches. Comment threads are displayed inline, and code changes are displayed in either an unified or split file view. Bitbucket implementation of the split file view illustrates where the new code lines are inserted in the original code, and where the removed lines would have been in the new file (Figure 2.9).

By connecting a repository to another tool by Atlassian - Crucible, the user can access a report on review coverage to determine if some parts of the source code is not covered by reviews, potentially exposing weaknesses in the code. Bitbucket’s premium tier subscription enables the team to configure what they call ”merge conditions”, which for example can require a set number of reviewers to approve the CR before the merging of code can happen (Atlassian, 2021b). As seen in top right corner of Figure 2.10, the merge conditions are listed so it’s clear what remains before the code merge can go through. To ease the workflow, Bitbucket has integration with Visual Studio code through an extension. In addition to other functionalities, this extension enables an author to create a pull request (change review) and the reviewers to review the pull request all inside their IDE (Atlassian, 2021c).

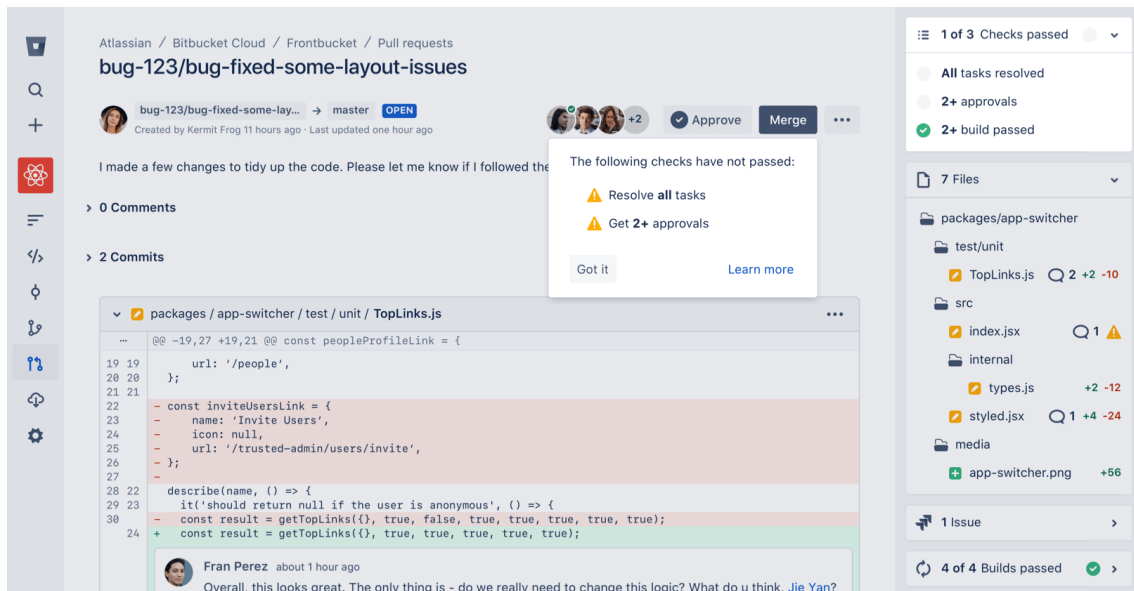


Figure 2.10: Bitbucket: Merge conditions

2.7.5 Review Board

Review Board is another web based CR-tool. From their website they claim to be used by big companies such as LinkedIn, Cisco, Twitter and many others (Beanbag, Inc., 2021a). The author can upload attachment files along their code, all files types are supported including multiple text and picture formats, PDF and audio. Files of these types can be reviewed, but Review Board does not support change tracking for binary file types (Beanbag, Inc., 2021b). The code changes are presented in a diff view of both the old and new version. Deleted code is marked with red in the old file, Added code is marked as green. Review board does also have a yellow marking, which is code lines that are modified. Another feature in the diff files, is that code that is not changed, but moved to another place in the code file will have a flag with the new code line position besides it in the old file view and from where in the new file view. This "moved flag" is a feature not seen in any of the other studied tools. Reviewers add comments by clicking either a line number or drag-select multiple code numbers. The comments support markdown, and can hence be used to make richer comments (hyperlinks, syntax highlighted code blocks, etc.) Comments from the reviewers are accessible from the diffs view, but by switching to the "reviews" view, the comment threads are displayed in a more neat way, with only the code changes and belonging comments taking the focus.

<pre> 175 """ 176 old_value = "" 177 new_value = "" 178 if 'old' in info: 179 old_value = info['old'][0] 180 181 if 'new' in info: 182 new_value = info['new'][0] 183 184 s = ['<table class="changed">'] 185 186 def render_change_entry_removed_value_html(187 info, old_value): 188 s.append(self.render_change_entry_removed_value_html(189 info, old_value)) 190 191 if new_value: 192 s.append(self.render_change_entry_added_value_html(193 info, new_value)) 194 s.append('</table>') 195 return ''.join(s) 196 197 def render_change_entry_added_value_html(self, info, value): 198 value_html = self.render_change_entry_value_html(info, value) 199 if value_html: 200 return ('<tr class="new-value"><th class="marker"></th>' 201 '<td class="value"><td></tr>' % value_html) 202 else: 203 return '' 204 def render_change_entry_removed_value_html(self, info, value): 205 value_html = self.render_change_entry_value_html(info, value) 206 207 208 209 210 </pre>	<pre> 175 """ 176 s = ['<table class="changed">'] 177 178 if 'old' in info: 179 old_value = info['old'][0] 180 181 182 183 184 if 'new' in info: 185 new_value = info['new'][0] 186 187 s.append(self.render_change_entry_removed_value_html(188 info, old_value)) 189 190 if new_value: 191 s.append(self.render_change_entry_added_value_html(192 info, new_value)) 193 s.append('</table>') 194 return ''.join(s) 195 196 def render_change_entry_added_value_html(self, info, value): 197 if value: 198 value_html = self.render_change_entry_value_html(info, value) 199 if value_html: 200 return ('<tr class="new-value"><th class="marker"></th>' 201 '<td class="value"><td></tr>' % value_html) 202 else: 203 return '' 204 def render_change_entry_removed_value_html(self, info, value): 205 if value: 206 value_html = self.render_change_entry_value_html(info, value) 207 </pre>
---	---

Figure 2.11: Review Board: Review interface

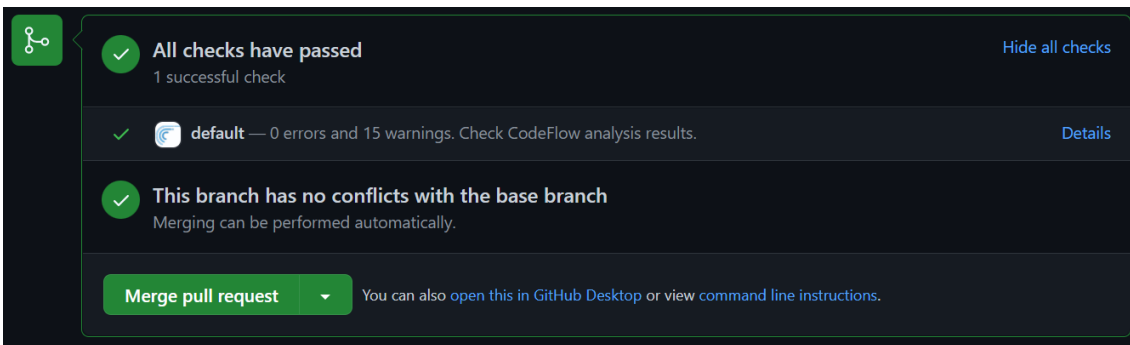


Figure 2.12: CodeFlow: Code quality feedback showing up in the GitHub pipeline

2.7.6 CodeFlow

CodeFlow is an example of a linting tool which support multiple languages. Being a linting tool means that it is able to identify code errors, exercise language specific code conventions and analyse the complexity of source code. Among the supported languages are the ones used in the courses in our scope (CodeFlow, 2021). CodeFlow has integration with GitLab, GitHub and Bitbucket, and can added to their CI-pipeline. This leads to the results from the code quality analysis showing up in the CI-pipeline section of the pull request, the results can by inspected on CodeFlow’s website by clicking the CodeFlow icon in the pipeline. As Bird and Bacchelli writes in *‘Expectations, Outcomes, and Challenges of Modern Code Review’*, with the support of automatic code analysis tools, the spared review resources can rather be used to review other parts of the code projects, which can bring more value.

```

9  def check(record):
    for field in fields:
11     if (field not in record):
12         return False
13         return check_valid_values(record)
14
15  def check_valid_values(record):
    record_pairs = record.split() # ['byr:123', 'iyr:24']
17     for pair in record_pairs:
18         att, value = pair.split(':')
19         if(att == 'byr'):
20             if(int(value) not in range(1920, 2003)):

```

 Missing function or method docstring missing-function-docstring

 Unnecessary parens after 'if' keyword superfluous-parens

 Missing function or method docstring missing-function-docstring

 Too many return statements (11/6) too-many-return-statements

 Too many branches (19/12) too-many-branches

 Trailing whitespace trailing-whitespace

 Unnecessary parens after 'if' keyword superfluous-parens

 Unnecessary parens after 'if' keyword superfluous-parens

Figure 2.13: CodeFlow: Code quality overview at their web page

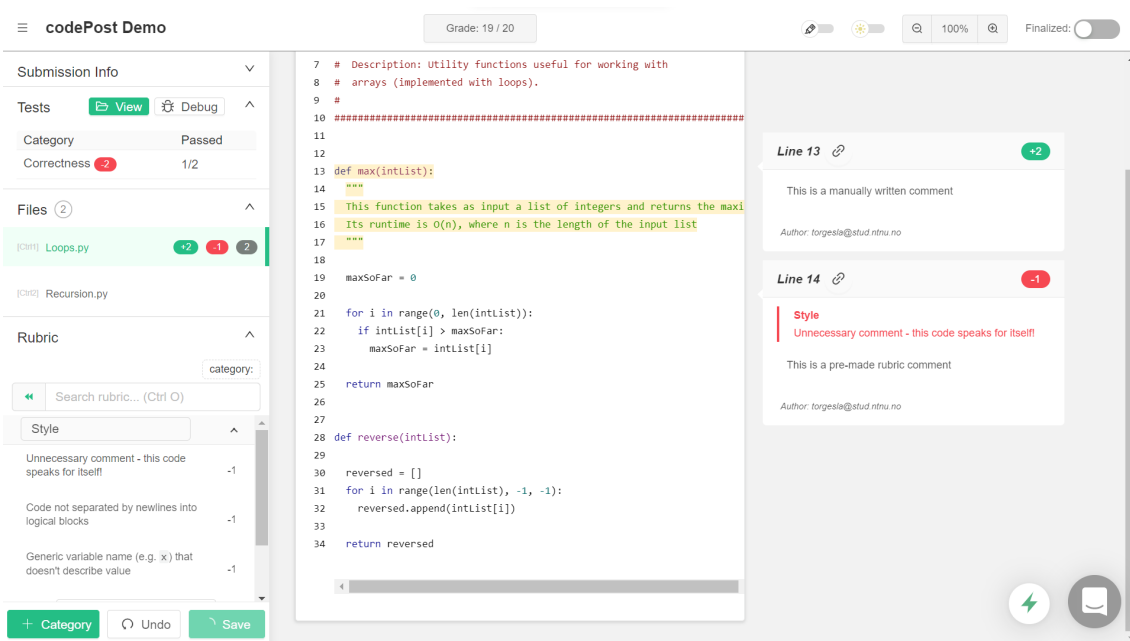


Figure 2.14: CodePost: Review interface

2.8 Features from educational tools

2.8.1 CodePost

The CodePost tool (CodePost, 2021a) is a web based CR-tool developed for an educational context. The goal of the tool is to improve feedback quality for the students and lower response time from a student work is submitted until it's assessed. The use case is similar to our use case, and is hence a tool to naturally take inspiration from. The students can submit their work through the platform directly, but the tool does also have integration with the biggest learning management systems (including Blackboard, Canvas and Moodle), anti plagiarism tools, GitHub, etc. through an API ((CodePost, 2021a) (CodePost, 2021b)). The main features of codePost is the commenting. The comment functionality is unlike the ones found in most of the CR-tools for the professional IT industry, besides Collaborator. The comment feature comes closer to the comment feature in Google docs. The comments are displayed in a panel on the right side of the code file. Comments can be attached to a part of the code by marking it with the cursor. The reviewer can then decide if the comment should affect the scoring points for the submission. As part of the autograder functionality of codePost, the teaching staff has the opportunity to write unit tests which will automatically run for every student submission, these can be set up to deduct or add points dependant on the correctness and efficiency of the

the biggest learning **management** systems (including Blackboard, Canvas and Moodle), anti plagiarism tools, Github, etc. through an API ((CodePost 2021a) (CodePost 2021b)). The main **features** of codePost is the commenting. The comment functionality is unlike the ones found in most of the code review tools for the professional IT industry, besides Collaborator. The comment feature comes closer to the comment feature in Google docs (**googledocs**). The comments are displayed in a panel on the right side of the code file. Comments can be attached to a part of the code by marking it with the cursor. The reviewer can then decide if the comment should affect the scoring points for the submission. As part of the autograder functionality of codePost, the teaching staff has the opportunity to write unit tests which will automatically run for every student submission, these can be set up to deduct or add points **dependant** on the correctness and efficiency of the code. To further streamline the review **process**, CodePost has the ability to create **rubrics** - reusable comments that can have a pre-set deduction or addition of scoring points, these are accessible through a hot key combination which makes them time saving versus writing them manually each time. The **rubrics** **does** also make it fair to the students, since a mistake of the same type and severity will deduct the same amount of

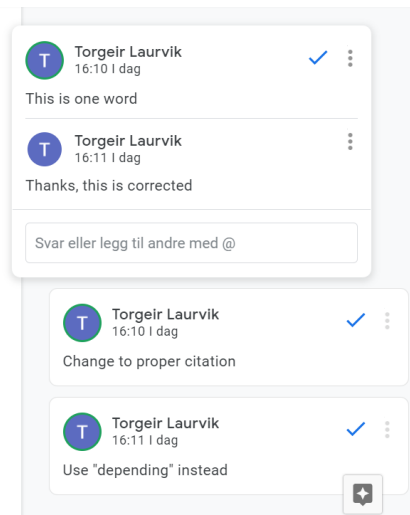


Figure 2.15: Google Docs: Comment feature

code. To further streamline the review process, CodePost has the ability to create **rubrics** - reusable comments that can have a pre-set deduction or addition of scoring points, these are accessible through a hot-key combination which makes them time saving versus writing them manually each time. The **rubrics** does also make it fair to the students, since a mistake of the same type and severity will deduct the same amount of points for every student. If the instructor decides to change the scoring point changing effect a rubric, the amount of points deducted or added will automatically be updated for every submission the rubric was applied to. As a last feature for directing the reviewer's focus, is that skeleton code handed out is de-emphasised in the review interface. This is hence CodePost's implementation of the industrial colorized diffs (e.g Figure 2.9) adapted for an educational use case.

2.8.2 HTML-report generated based on tests - currently in use

The HTML reports are the support tool currently in use for reviewing of exams in several of the bachelor programming courses at NTNU. This HTML-based solution was developed by a research assistant at NTNU, and was suppose to be used temporary until a more robust solution was ready. It brought value in form of streamlining the review process, and has hence been kept used. The value in the solution does mainly come from the results from automatically ran unit tests, which efficiently can tell the sensor what parts of the code that needs more attention and the ones that passed the unit tests might need less attention. The tests are weighted with a point

```

public static void Printable(String table) throws SQLException {
    String query = "select* from "+table;
    ResultSet result = Read(query);
    int columns = result.getMetaData().getColumnCount();
    while(result.next()) {
        String string = "";
        for (int i = 1; i < columns+1; i++) {
            string+=result.getString(i)+",";
        }
        System.out.println(string);
    }
}

public static void PrintSet(ResultSet result) throws SQLException {
    int columnCount = result.getMetaData().getColumnCount();
    while(result.next()) {
        String string = "";
        for (int i = 1; i < columnCount+1; i++) {
            string+=result.getString(i)+",";
        }
        System.out.println(string);
    }
}
}

```



Figure 2.16: CodePost: De-emphasized template code

score, the point score for each block of tests is then summarised and displayed to the reviewer next to the file-in-review. To further streamline the code inspection, the solution highlights the code lines added to the skeleton code written by the course staff. This helps the reviewer by directing their focus to the actual student written code. Both the source code files written by the student and the test files written by the course staff are presented in a list view which is possible to scroll through. This makes it easy to switch back and forth between multiple files. The test files are by default collapsed, but by being available, the reviewer is able to check why a test failed if it's not immediately clear just by reading the source code.

2.8.3 CodeTour

CodeTour is an extension to the code editor, Visual Studio Code (Jonathan Carter, 2021). Its intended use is to record guided tours of your code base, to in a better way introduce or re-introduce a person to an unfamiliar code base. The user clicks the line number(s) of a code snippet, it wants to explain, in its code file, and do then write the comment explaining the code. In our use case, these attached comment can rather be used by the reviewer to give the author feedback. The comments allows markdown, and can there be used to explain the system in a richer way than what simple comments could have done. The tour part of CodeTour is meant to help the unfamiliar user get a though through introduction to the code rather than exploring the code in a "blind way" through documentation, docstrings and regular

DelegatingTemperature.java

```
package del4;

public class DelegatingTemperature {
    // Add any needed fields
    private Temperature delegate;
    private char scale;
    /**
     * @param delegate - the Temperature to get degree from
     * @param scale a character declaring the scale of this
     temperature object
     *
     * @throws IllegalArgumentException if scale is not 'C' or 'F'
     */
    public DelegatingTemperature(Temperature delegate, char scale) {
        // TODO
        if (scale != 'C' && scale != 'F') {
            throw new IllegalArgumentException("The scale must be
either Fahrenheit or Celcius!");
        }
        this.delegate = delegate;
        this.scale = scale;
    }

    /**
     *
     * @return The current scale
     */
    public char getScale() {
        // TODO
        return this.scale;
    }
}

del4.DelegatingTemperatureTest

• OK - 1.0 - testDelegatingToOtherReturnsCurrent
• OK - 0.5 - testInvalidScaleThrows
• OK - 1.5 - testGetDegreeIsCorrectInSameScale
• OK - 1.0 - testDelegatingInOtherDoesNotModifyCurrent
• FAILURE - 1.0 - testDelegatingInOtherReturnsCorrectValues
  org.opentest4j.AssertionFailedError: expected: <50.0> but was: <10.0>
    at
    del4.DelegatingTemperatureTest.testDelegatingInOtherReturnsCorrectValu

• FAILURE - 1.0 - testDelegatingToOtherReturnsCorrectValue
  org.opentest4j.AssertionFailedError: expected: <50.0> but was: <10.0>
    at
    del4.DelegatingTemperatureTest.testDelegatingToOtherReturnsCorrectValu

• FAILURE - 1.0 - testInOtherAndToOtherDoesNotModifyDelegate
  org.opentest4j.AssertionFailedError: expected: <50.0> but was: <10.0>
    at
    del4.DelegatingTemperatureTest.testInOtherAndToOtherDoesNotModifyDeleg

• FAILURE - 1.0 - testDelegatingGetDegreeIsCorrectInOtherScale
  org.opentest4j.AssertionFailedError: expected: <50.0> but was: <10.0>
    at
    del4.DelegatingTemperatureTest.testDelegatingGetDegreeIsCorrectInOther

4.0 of 8.0 points
```

Figure 2.17: HTML-tool: Student written code and test results

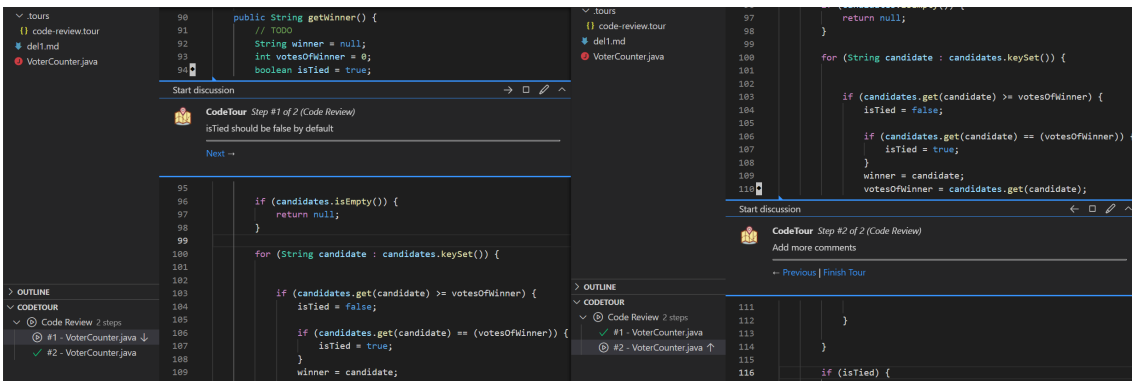


Figure 2.18: CodeTour with added review comments

comments.

Some of the value of CodeTour as a CR-tool is that it is lightweight and easy to use. Both the reviewer and author can do the CR without leaving their code editor. The review comments are saved in a separate folder inside the project folder and can easily be shared with the students or other reviewers. The code tour comments/steps can be interactive, and can even include code that once clicked will be ran. Although CodeTour has never been used as a assessment tool at NTNU neither as a CR-tool in the industry, we decided to include it as a tool for inspiration, as there are similarities between our use case and CodeTour's intended use.

Features \ Tools	Gitlab	Github	Collaborator	Bitbucket	Review Board	CodePost	HTML-based	CodeTour
Red and green diff view	x	x	x	x	x	De-emphasizes template code	x	
Yellow highlighting for changed line			x		x			
Flag for moved code					x			
Split diff	x	x	x	x	x			
Unified diff	x	x	x	x	x	x	x	
Inline comments	x	x						x, one CodeTour-step displayed at a time
Different comment types			x					
Comments in side pane			x			x		
React to comment	x	x		x				
Automatically run tests	x	x				x		
Report on review coverage			x					
Web based	x	x			x	x		
Code review in IDE	x	x	x	x				x
Show files in different pages	x							
Code quality indicators in review view	x							
Finished review checkbox on each file	x				External CI-tools			
Display documentation files	Image and PDF	Image and PDF	x	Image and PDF (through add-on)	x	Image and PDF		
Diff tracking documentation files			x		x			
Metrics about the reviewers			x	Maybe				
Integration with source code hosting service	x	x	x	x	x	x		
Markdown in comments	x	x	x	x	x	x	x	
Integration with LMS								
Integration with anti-plagiarism tools						x		
Attach assignment point score effect to comment						x		
Result from tests affect point score						x	x	
Reusable comments (rubrics)						x		

Figure 2.19: Overview of features from the studied tools

2.9 Why are there differences between the educational and professional tools?

Both categories of tools is developed specifically for their respective use case. Hence, differences between tools made for a educational use case and the professional tools arise as a result of the motivations behind the use of each tool and the form of the code-in-review, as explored in Section 2.6.

The motivations between CR in the educational and industrial setting are different by overlapping. As we saw from Section 2.5, there are multiple motivations for conducting CRs in the industry. The results from the research by Bird and Bacchelli, 2013 even showed that the primary usage of CRs in that department was not for the reasons that developers and managers had expected and stated as their main motivation. To sum up the motivations, they were mostly focused on improving the product or code in different ways - through correcting defects, correcting bad code or suggesting better alternative solutions. Although not being the top motivation for almost any of the interviewed (only 8%), everyone (except one) did mention learning or knowledge transfer as a motivation. Since preventing defects and bad code to sneak into the code base is the number one priority, and the features of such tools focus on making this less likely to happen. The code is inspected thoroughly, and the reviewers and author must have a forum to discuss the changes proposed by the author.

There are even internal differences in the educational tool, between the tools intended for reviewing of formative assessments and those intended for summative assessments. In formative assessments, the learning outcomes or knowledge transfer is main motivation (Helle and Burner, 2021), and hence is this reflected in the tools intended for this kind of usage. The reviewer can still benefit from tools that make it easier to identify defects and bad code, but mainly to have concrete examples from the submission to base feedback on. CodePost is an example of a tool made with the formative assessments in mind. The tool is made *"to help you give amazing feedback, quickly"* (CodePost, 2021a), this means that the quality of feedback is the main focus of this tool. It does also lead to a streamlining effect, which is sought after for both formative and summative assessments.

For the final exam and other summative assessments, the situation is a bit different. The knowledge transfer effect is not the main motivation anymore. For this kind of use case, the focus is rather on a tool that can streamline the review process.

Educational tools have a differing relative focus between these two motivations, leading to the internal differences between them. For example, the HTML-based tool was specifically made with reviewing of the final exam in mind. Based on unit tests, the tool reports to the reviewer sections of the code that didn't behave as expected, so that the reviewer can prioritize inspecting this faulty code and not the code that fulfilled the unit test requirements. The tool therefore emphasizes different functions than CodePost.

Another key difference between educational and industrial tools are the form of the code in review. In the industry, existing code is updated in small increments and reviews are focused on the latest updates of the code. The code-in-review is therefore usually a small amount of code, and the need is for features that allow thorough and ongoing discussion of few lines involving multiple reviewers. On the other hand, the educational assessment tools analysed for this thesis has mainly been designed for use by one reviewer at a time, concerning larger blocks of code. The inline discussion thread functionality may therefore be unnecessary for the educational tool. Furthermore, since reviewers have to inspect large blocks of code, features that allow the creation of inline discussion threads which visually split up the code may prove disruptive for keeping an overview of the whole code.

However, with the new change of law (Kunnskapsdepartementet, 2005), the need for a communication channel for the two reviewers inside the review tool will emerge, and hence a feature that mainly was relevant for the industrial tools will also be more relevant for our use case. Since the change of law is exclusive to Norway, it is unlikely that this change of law will change the educational tools as they are not targeted specifically at the Norwegian market.

Chapter 3

Feature Selection

The previous sections laid out a thorough mapping of the use cases of CRs in both the industry and the educational setting, as well as the resultant features of the tools currently in use in both settings. Based on this analysis, the features of current CR-tools used in the industry which are most relevant for our education use case are identified and outlined in the following section.

3.1 Priority of courses

3.1.1 Web development

When assessing the structure of the web development course (Section 2.3.4), it becomes clear that this course bears significant differences compared to the other courses in scope.

First, programming has traditionally not been part of the exam in the web development course, while for the other four courses, programming is the main component. This thesis' objective is to streamline the reviewing of programming submissions. The exam form of the web development course therefore falls outside the objective of the thesis. However, the several development projects throughout the course semester are still a relevant use case for a specialised review tool.

Second, even the projects in the web development course which may be a relevant use case of a review tool have a different focus than the submissions for the other courses. In these projects, it is the behavioral requirements of the student-written

program which is in review ((GeeksforGeeks, 2019), (Appendix C)), while in the other courses, the focus is on streamlining the inspection process of the concrete source code that students write. Hence these features are not as relevant even for these projects.

Third, for the other four courses, the submissions are reviewed by senior staff and TAs, while for the web development course, the wish is for a review system where the students are the main users, and which facilitates student active assessment forms (Appendix C).

Given these three differences between the web development course and the other four courses, we believe that including this course's requirements and needs would pull the design in a direction that would compromise its usefulness for both use cases. Hence, we have decided to rather prioritise the requirements in the other four courses in our first conceptualization of the tool, upon which future work can build to include the requirements of the web development course. Based on what T. Aalberg writes about how peer reviews are conducted as of today's practice in the course, it is clear that a specially made review tool is beneficial for this use case.

3.1.2 Introduction courses (ITGK and OOP) and Software development course

The submissions in the software development course have a closer resemblance to the source code projects in the industry, for which there already exists a lot of high quality CR-tools, some of which are discussed in this thesis. The courses based on Python, Java and Python on the other hand, where the potential for streamlining is the largest, as they have a structure which make it possible to use unit tests to automate large fractions of the review work, is not covered by existing tools. These three courses are the only courses with a programming based exam, which we based on the existing practice in today's tool (Section 2.8.2), the recommendation from Bird and Bacchelli and the autograder features of CodePost being one of its main selling points as a review tool for the educational use context. Based on the lower existing supply of tools designed for these three courses than tools useful for the software development course, it is natural to give the feature requirements from ITGK and OOP a higher priority than the ones from software development.

3.2 Industry-specific features

Since the majority of the reviewed tools are made for use in the professional industry, we expect several of the features they implement are industry-specific features, some of which are unnecessary or even counterproductive for our tool to adopt. Through the background we have mapped out the similarities and differences in motivations and use case. In *Why are there differences between the educational and professional tools?*, we have even seen what effect these differences have made on the tools made for each of the two use cases. Some industry-specific features can be relevant for our use case, even though their original implementation might not be the correct one for our use case, these will be discussed in the next section. With web development removed from the courses effecting the feature selection, the users for our system are primarily senior staff and some students through the role of teaching assistants. Features regarding supervision of the review process is therefore not relevant for our use case. These features include *"Metrics about the reviewers"* and *"Report on review coverage"*.

As a common use case for the industrial CR tools are in connection to pull requests, the reviewer can decide to not approve this request. The author then suggest a new version of its contribution with improvements based on the reviewers remarks for why the pull request was not approved. This loop will continue until the reviewers are happy and approved the code merging. This approval loop is not present in our educational use case. A submission is submitted and then assessed. The features regarding approval of a review / pull request is hence not relevant for our use case. These features are the multiple comment types from Collaborator and the alternative way reviews are approved or not approved in the other tools.

3.3 Selection criteria for features

The motivations for making such a tool is mainly to streamline the review process. As a result of the change of law taking effect from 1. august 2022 (Kunnskapsdepartementet, 2005), the need for a reviewing tool which enables the reviewers to collaborate is even more pressing. The current HTML-based review solution in use in several of the bachelor programming courses does not facilitate any form of collaborative review or shared comment threads. The current solution is also lacking a way to write the review comment in the same application used to view the code

submission. This means that with the solution currently used today, the reviewer's comments are not visually connected to the code. By improving upon this, a greater learning effect might be achieved.

Since there are limited resources available to realise the review tool proposal from this master project, it makes sense prioritise the most important features for our use case. While a long list of features might be potentially useful for our use case, the resource cost to implement them may actually be greater than the benefit. It is therefore critical to identify which features are must-haves, and which others are just nice to have.

The selected features must:

1. Enable multiple reviewers to collaborate on the review
2. Write comments in the same view as the code is inspected
3. Streamline the review process

3.4 Discussion of each feature

The selection of features for the final design proposal will be based on whether a feature contributes to the system's three functional requirements. Both the first and second requirement are possible to achieve just by making the choice of implementing functionality which enables them, as neither of them are even present in today's support tool. The third requirement on the other hand is more open-ended, and does always have room for improvement. We know that we want to streamline the review process to be able to prioritise parts of the review which brings more value, but the resources for developing a new tool is limited and each suggestion should be considered based on cost / benefit.

3.4.1 "Enable multiple reviewers to collaborate on the review"

Collaboration on reviews as a feature is present in all of industrial CR-tools. This means that the system must at least implement comment threads. Implementation of a voting system in addition might contribute to keeping the interface clear as

opposed to the alternative which is a second reviewer showing support through a comment.

3.4.1.1 Voting system

The motivation behind implementing an up- and downvote system as a feature is to avoid duplicate comments and comments which just contains approval. This leads to a cleaner interface for all of the users while giving the reviewers a quick way to show approval or disagreement. Such a feature is a nice to have feature, but based on the simplicity of implementing it, it should be considered.

3.4.2 "Write comments in the same view as the code is inspected"

To meet this requirement, there is no concrete implementation choices to make other than deciding to adopt a feature found in all the systems discussed in this thesis except CodeFlow and the current system - the one we aim to improve upon. The question is rather how we want to implement it.

The features connected to comments seen from the discussed tools are:

- Comment inline or in side pane
- Ability to attach point score to a comment
- Reusable comments - rubrics

3.4.2.1 Comments inline or in side pane

As described in Section 2.9, in our use case, we do not expect long discussions for a single block of code. The need is rather for a comment feature that makes it easy to keep overview of the submitted code without having to navigate a lot. Based on this characteristic in our use case, it is natural to prefer a side pane comment feature as it is implemented in Smartbear Collaborator (Section 2.7.3) and CodePost (Section 2.8.1), rather than the inline comment feature found in most of the other review tool. The comment feature must support threads where multiple reviewers can communicate, as the tool is required to implement features

which *"Enable multiple reviewers to collaborate on the review"*. A thing to consider is that by implementing a side pane comment view, the application will increase in width, which can result in less space for the source code view if the user's screen is too narrow.

3.4.2.2 Ability to attach point score to a comment

This feature is connected to *Propose of grade based on test scores* and does make sense to implement together. This is the approach taken by codePost, it takes away the need for the reviewer to keep track of the sum of points, as this is automatically calculated by the system. This features does also make it possible for the review to give partial score for task even if the belonging unit test(s) failed.

3.4.2.3 Reusable comments - rubrics

As many of the students do similar or identical mistakes, the review comments will also be reused. By borrowing the implementation of a "bank" of reusable comments, this will streamline the writing of comments, freeing reviewer time. It does also make it easier to show relevant examples or link the student to existing learning resources which will increase the value of the formative for the formative assessments as well. Connected to the latter paragraph, if the reusable comments have an attached point score it will make it easier for the reviewer to be fair in regards to deducting the same amount of point for the same mistake for each student.

3.4.3 "Streamline the review process"

3.4.3.1 Integration with tests

In the HTML-based tool currently in use at NTNU (Section 2.8.2), the feedback from the automatically ran unit tests is the main feature, and has a great impact one the efficiency of the review. As Bird and Bacchelli writes in their paper, by automating the parts of review process which doesn't require human understanding, the scarce review resources can rather be used to improve the formative assessment of the students by reviewing the deeper defects. It does there makes sense to implement feedback from the unit tests in our tool as well. Especially for the introduction course, it can make great impact to implement some degree of automation of the

review through the use of unit tests, as the review is done manually as of today.

3.4.3.2 Highlighting in diff

The use of colorized diff in the industrial tools is present since it is only the new proposed changes that are to be reviewed. The use of colors directs the reviewers focus to the code-in-review, and away from the code that is not relevant for that particular review. In our use case, the case is similar, the use of colors directs the reviewers focus. The use of de-emphasising of handed out skeleton code in codePost and highlighting in the current HTML-based solution, which is both designed with an educational use case in mind, does further emphasise that such a feature is valuable in order to streamline the review process for our use case.

De-emphasising of code in CodePost CodePost has taken another approach than the other tools for guiding the reviewers focus to the changed lines of code in the industrial tools or the student-written blocks of code in the educational tools. While the other tools highlight the important code, CodePost de-emphasises the code skeleton by making it contrasting less to the background than the code which should get the focus. As we can see from Figure 2.16, the skeleton code is barely visible both for light and dark mode. This is a problem, as it is important for the reviewer to have the ability to quickly identify what function the student code belongs to. Our assessment is hence that this feature can be counterproductive in regards to streamlining, and that highlighting of the code-in-review is the better approach.

In the researched industrial tools, the source code is constantly developing both in the added and removed dimension from review to review. For those projects, it therefore makes sense to use different colors of highlighting to make the changes clear.

The submissions in the introduction course to programming and the two object-oriented programming courses do take the form of student code added to a skeleton. This means that code is never removed, just added, hence do we only need one color for highlighting the additions. All of the work on the submission is done between the time of tasks being handed out and the final review of the submission.

As mentioned in Section 2.6.2, we might encounter some submissions that have been reviewed before and then revised before a new review is conducted. This means that the use of multiple colors for highlighting of changes is relevant for these submissions.

By using a color for deleted code and one for added code, we can also make it easy to identify the unexpected case of a student deleting code from the code skeleton, as the reviewer of the submissions based on code skeletons is not suppose to encounter any code highlighted as deleted. The decision is hence to implement highlighting of added and deleted code lines.

3.4.3.3 Split and unified file view?

Another consequence of how the source code for most submissions goes from just consisting of the code skeleton at the time of hand out to the finished state at the time of review, is that it doesn't make sense to consider a split view for inspecting the changes. The "old code" view does only serve a purpose when code is removed from the old code version, while for our use case, this behaviour is not to be expected, as removed code will mean that the student has modified the function signature in the code skeleton. For the revised submissions, where changes build upon the last reviewed version, the split file view can make sense to implement as a choice for the user based on user preference. As the unified file view is capable of presenting the types of changes for both types of submissions, the split file view is considered a nice-to-have feature, giving it a lower implementation priority.

3.4.3.4 Propose of grade based on test scores

The wish for a system which proposes a grade based on the unit tests was originally expressed by the C++ course staff. Such a feature does also make sense for both ITGK and the Java-course, as both of these courses have submission which are easy to review using unit tests. In an implementation where the system keeps track of the scoring points, the proposal of a letter-grade is an addition which requires minimal effort.

3.4.3.5 Linting tool feedback in the review view

As written in Section 2.9, the stylistic and code convention errors is not the knowledge-in-review in the courses in our scope. Linting tools, such as CodePost (Section 2.8.1), are though in addition able to detect the code line causing a defect, and having them as support tool can hence be effective to cut down the time it takes to identify the defect causing a unit test to fail, thus streamline the review process. To reduce the

number of elements fighting for the reviewers attention on screen, an idea can be to make it optional to show these indicators. This option should be easy to toggle on and off.

Chapter 4

Implementation considerations

To give a mental picture of how the system can look, we have made a mock-up of a tool implementing some features from the backlog. The decision to display the multiple supporting feature alongside the source code view, is for the reviewer to be able view larger portions of the source code without have to zoom out or scroll, as features taking up vertical screen space covers potential code-showing space. These features include comment threads, code quality indicators, results from unit tests and the rubrics comment menu. This means that it becomes important to reduce the horizontal screen space occupied by these elements. Reflecting on the usage of such a tool, it become apparent that one way to reduce the number of elements on screen at a time is through switching between what elements are shown. The unit test results are useful for directing the reviewer to what code needs a more thorough inspection, but might not have to visually accessible all the time. When the reviewer starts the review, it is the unit test-pane which is open, ready for the guiding the reviewers focus to the When the reviewer has found code to comment on, the comment view is toggled by a hotkey (or switching by pressing the button) to show, and the unit test view is hidden. From Figure 4.1 we can see how the use of bright colors to show the result of the tests is telling the reviewer what code failed the unit tests, and which one that didn't. Each test is worth 10 points, so the total scoring points for this submission is 10 points before the further inspection of the code is done manually. The total score is displayed inside the *"Finish review"*-button above the test result pane. We can also see from this figure, that all lines are modified except for the function signature on line 8.

By switching on static code analysis annotations, we can see remarks on code showing up as small icons in the left margin between the line number and the source code

Code quality indicators (⚙️+F6)

Toggle comments/tests (⚙️+F7)

Text-chat

Save progress (⚙️+S)

Finish review Score: 18.5/20

- Submission
- task1.py
- task2.py
- task3.py

```

1 import sys, os, re
2
3 with open(os.path.join(sys.path[0], 'input.txt'), mode='r', encoding='utf-8') as datafile:
4     parantheses = datafile.read()
5
6 def floor(): return parantheses.count('(') - parantheses.count(')')
7
8 def pos_baseament():
9     current = 0
10    for i, char in enumerate(parantheses):
11        current += 1 if char == '(' else -1
12        if current == -1:
13            return i

```

Rubrics	Search for rubric (⚙️+R)
For-loop: Syntax Error	-1
Need for comment	-0.5
Indentation mistake	-0.5
Use of context-manager	2

Floor function test 10/10

Position baseament test 0/10

Figure 4.1: Mock-up: Showing the results from unit tests

Code quality indicators (⌘+F6)

Toggle comments/tests (⌘+F7)

Text-chat

Save progress (⌘+S)

Finish review

Submission

- task1.py
- task2.py
- task3.py

```

1 import sys, os, re
2 PEP-8: Imports
3 * Imports should usually be on separate lines:
4 # Correct:
5 import os
6 import sys
7
8 # Wrong:
9 import sys, os
10
11 https://www.python.org/dev/peps/pep-0008/#imports
12 if(current == -1):
13     return i

```

==> encoding='utf-8') as datafile:

s.count())

Floor function test 10/10

Position baseament test 0/10

Rubrics	Search for rubric (⌘+R)
For-loop: Syntax Error	-1
Need for comment	-0.5
Indentation mistake	-0.5
Use of built-ins	2

Figure 4.2: Mock-up: The annotation from the static code analysis tool on hover

Code quality indicators (⌘+F6)

Toggle comments/tests (⌘+F7)

Text-chat

Save progress (⌘+S)

Submission

- task1.py
- task2.py
- task3.py

```

1 import sys, os, re
2
3 with open(os.path.join(sys.path[0], 'input.txt'), mode='r', encoding='utf-8') as datafile:
4     parentheses = datafile.read()
5
6 def floor(): return parentheses.count('(') - parentheses.count(')')
7
8 def pos_basement():
9     current = 0
10    for i, char in enumerate(parentheses):
11        current += 1 if char == '(' else -1
12        if current == -1:
13            return i

```

Rubrics	Search for rubric (⌘+R)
For-loop: Syntax Error	-1
Need for comment	-0.5
Indentation mistake	-0.5
Use of context-manager	2

Finish review

Score: 18.5/20

Rather import in multiple lines -0.5

Use of context-manager 2

Write functions in multiple lines -0.5

Test#2 failing is due to a spelling mistake (enumerate), no lack of knowledge 8.0

Indentation mistake (after if) -0.5

Figure 4.3: Mock-up: Toggle to write comments view

line in line 1, 6 and 13 (Figure 4.2). By hovering over the indicator, the remark shows up with an explanation of the problem. In Figure 4.3, the reviewer has added five comments as the review of the submission. The commented code is highlighted in light blue. The total of points for the submission is by this raised to 18.5. The second comment is added by using one of the pre-made rubric comments.

4.1 Proposal for sprints

As discussed in Section 3.1.2, the impact of such a tool is greatest where it there exists no alternatives, we propose prioritising the tool for the three courses where the course staff has the advantage of knowing the function signatures for each of the implementation task.

4.1.1 Sprint 1

All the essential features must be implemented in sprint 1. Essential features are all the features required for the tool to be a better alternative to the existing review tools. By following the principle of *Minimum viable product* (Becker, 2020), the earliness of release of a better review tool (compared to today's solution) is prioritised. In addition to the reviewers having access to a viable product early, by choosing this approach, it also brings the advantage of the reviewers being able to influence the development of the tool in an early stage, guiding the further development in a desired direction.

The two functional requirements: "*Enable multiple reviewers to collaborate on the review*" and "*Write comments in the same view as the code is inspected*" must both be implemented in the first sprint for the tool to have an edge over the current review system. In addition, the presentation of feedback from unit tests and highlighting of the student written code should be added in sprint 1 as these are features present in today's solution.

- Comments inside the review view
- Collaboration functionality
- Tests results in review view
- Diff highlighting

4.1.2 Sprint 2

In this sprint, the features further streamlining the review should be implemented. This includes the scoring point tracking - based on tests and comments and the letter grade proposal - based on the total amount of scoring points.

- Scoring points for unit tests and comments
- Letter grade proposal
- Syntax highlighting

4.1.3 Sprint 3

In these sprints, the features based on requirements from the software development course should be prioritised. These include a flattened presentation of the directory tree as well as a student user account, as this is the first use case where the system must support both reviewer accounts and student accounts with their belonging rights and restrictions.

- Flatten file view (from GitLab)
- Checkbox for each file (from GitLab)
- Hide file-types and from what folder (course-specific preset)

4.1.4 Subsequent sprints

In the subsequent sprints, the remaining features are implemented. This includes the features the developer finds to be relevant in addition to:

- Code quality indicators
- Voting system for comments
- Integration with LMS and Git

Chapter 5

Discussion

5.1 Contribution

Based on the research done as background for this thesis, it became clear that the concept of using industrial CRs as inspiration for a educational review tool targeted at assessments was not covered in existing research. Most of the papers on the use of code review techniques in an educational context was concerned with peer review - where the students review each others submissions. This was a different use case than the one we had decided to explore.

This thesis serves to fill this gap in existing research in the field. The thesis' first contribution is gathering research on the motivations behind code reviews in the industry, existing practices for code reviews in the industry and in the educational setting.

Second, the thesis contributes by conducting a systematic comparison of the use case of educational code reviews and code reviews in the industry, subsequently mapping which features in industrial tools are of key relevance and suitable to be transferred to an educational tool.

As the final contribution, and the core goal of the thesis, we have proposed a concrete sprint-based plan for how the desired review tool should be developed to enable an early release of a viable product and maximize the usability gained from each sprint cycle.

5.2 Limitations

Information about most of the courses were gathered through direct correspondence with the course staff, with the exception of ITGK, where the author was unable to get in contact with the staff. In place of this, information about the review process the course was gathered from my supervisor and course information pages (NTNU, 2021d and Appendix D). The supervisor of this thesis is also the lecturer for TDT4100 (OOP using Java) and has been part the course staff for TDT4140 (Software development). The information for these courses are hence based on oral and written explanations from him.

5.2.1 Data base

Thesis' goal and domain is definition of functional requirements. As a result of this the knowledge background upon which the proposals are based, are less based on theoretical sources, and more on analyses of the underlying use case conditions, wishes from the course staff and by inspection of in total nine existing industrial and educational tools for inspection of source code.

5.3 Implications and future direction

As mentioned in Section 5.1, this exact application of knowledge transfer from the industrial code review practice to an educational non-peer review review setting had yet to be covered in existing research. This thesis serves as an introduction to an application where industry knowledge can make great impact.

For this thesis the focus has been on the streamlining effect the industry knowledge could deliver to this use case, but further research is encouraged to investigate other applications of the knowledge transfer from the industrial to the educational setting.

The concrete result of this thesis is the informed features proposal, and an order of priority of these features in the form of a sprint plan is based on the principle of minimum viable product (Becker, 2020). This ensures that the tool is useful in an early stage by prioritizing the features needed for the tool to be useful enough for the users to adopt it and it also leads to the developers having a source of feedback which can help to guide the further design of the system in a direction according to

the users wishes.

As there are many commercial actors in the industrial code review market, a collaboration between the industrial actors and educational institutions could be beneficial for both parties. Based on this thesis it has been proven that many of the features from the industrial tools are relevant for the educational setting despite the motivational and use case differences discussed in Section 2.6.

The lack of existing systems specifically aimed at streamlining the process of assessment of programming submissions, makes this a potentially valuable business case.

Bibliography

- Atlassian. (2021a). Retrieved 5th December 2021, from <https://bitbucket.org/>
- Atlassian. (2021b). Retrieved 30th November 2021, from <https://bitbucket.org/product/features/code-review>
- Atlassian. (2021c). Jira and Bitbucket (Atlassian Labs). Retrieved 30th November 2021, from <https://marketplace.visualstudio.com/items?itemName=Atlassian.atlascode>
- Beanbag, Inc. (2021a). Retrieved 3rd December 2021, from <https://www.reviewboard.org/>
- Beanbag, Inc. (2021b). Retrieved 29th November 2021, from <https://www.reviewboard.org/#slide-pdf-review>
- Becker, R. (2020, August 14). Minimum viable product (mvp) (Techopedia, Ed.). <https://www.techopedia.com/definition/27809/minimum-viable-product-mvp>
- Bird, C. & Bacchelli, A. (2013). Expectations, outcomes, and challenges of modern code review (Proceedings of the International Conference on Software Engineering). *Proceedings of the International Conference on Software Engineering*. <https://www.microsoft.com/en-us/research/publication/expectations-outcomes-and-challenges-of-modern-code-review/>
- Bradford, D. L. (2019). Ethical Issues in Experiential Learning. *Journal of Management Education*, 43(1), 89–98. <https://doi.org/10.1177/1052562918807500>
- Burner, T., Baraas, R. & Falkenberg, H. (2011). Studentaktive vurderingsformer i norsk lærer- og optometriutdanning. *University Pedagogics*, 1, 44–57. <https://doi.org/10.18261/ISSN1893-8981-2011-01-04>
- CodeFlow. (2021). Retrieved 8th December 2021, from <https://www.getcodeflow.com/#supported-platforms>
- CodePost. (2021a). Retrieved 18th November 2021, from <https://codepost.io/>
- CodePost. (2021b). Retrieved 18th November 2021, from <https://github.com/codepost-io>

-
- Cohen, J. (2006). *Five types of review*. "Smart Bear Inc." Retrieved 4th December 2021, from <https://www.ccs.neu.edu/home/lieber/courses/cs4500/f07/lectures/code-review-types.pdf>
- Eduflow. (2021). Empower students to learn collaboratively. Retrieved 17th November 2021, from <https://www.eduflow.com/higher-education>
- GeeksforGeeks. (2019, August 9). Retrieved 9th December 2021, from <https://www.geeksforgeeks.org/system-testing/>
- GitHub. (2021). Retrieved 5th December 2021, from <https://github.com/#home-collaborate>
- Github. (2021). Retrieved 13th December 2021, from <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/collaborating-on-repositories-with-code-quality-features/about-status-checks#checks>
- GitLab. (2021). What are the most effective features for code review tools? Retrieved 22nd November 2021, from <https://about.gitlab.com/topics/version-control/what-are-best-code-review-tools-features/>
- Gitpod. (2021). Retrieved 17th November 2021, from <https://www.gitpod.io/docs>
- Helle, L. (2020). Summativ vurdering. https://snl.no/summativ_vurdering
- Helle, L. & Burner, T. (2021). Formativ vurdering. https://snl.no/formativ_vurdering
- Jonathan Carter. (2021). Codetour. Retrieved 25th November 2021, from <https://github.com/microsoft/codetour>
- Kolb, D. A. (2014). *Experiential learning: Experience as the source of learning and development*. FT press.
- Kunnskapsdepartementet. (2005). Lov om universiteter og høyskoler (universitets- og høyskoleloven). https://lovdata.no/dokument/NL/lov/2005-04-01-15/KAPITTEL_1-5#%C2%A75-3
- NTNU. (2021a). IT2810 - Web Development. Retrieved 17th November 2021, from <https://www.ntnu.edu/studies/courses/IT2810>
- NTNU. (2021b). TDT4100 - Object-Oriented Programming. Retrieved 9th November 2021, from <https://www.ntnu.edu/studies/courses/TDT4100>
- NTNU. (2021c). TDT4102 - Procedural and Object-Oriented Programming. Retrieved 9th November 2021, from <https://www.ntnu.edu/studies/courses/TDT4102>
- NTNU. (2021d). TDT4110 - Information Technology, Introduction. Retrieved 9th November 2021, from <https://www.ntnu.edu/studies/courses/TDT4110>
- NTNU. (2021e). TDT4140 - Software Engineering. Retrieved 9th November 2021, from <https://www.ntnu.edu/studies/courses/TDT4140>

-
- O’Leary, B. (2021). How gitlab’s 5 new code review features will make life easier. Retrieved 22nd November 2021, from <https://about.gitlab.com/blog/2021/09/09/5-code-review-features/>
- Radigan, D. (2021). Why code reviews matter (and actually save time!) Retrieved 6th December 2021, from <https://www.atlassian.com/agile/software-development/code-reviews>
- Silverthorne, V. (2021). The code review struggle is real. here’s what you need to know. <https://about.gitlab.com/blog/2021/09/03/the-code-review-struggle-is-real-heres-what-you-need-to-know/>
- Smartbear Software. (2021a). Retrieved 10th December 2021, from <https://smartbear.com/product/collaborator/overview/>
- Smartbear Software. (2021b). Retrieved 5th December 2021, from <https://smartbear.com/product/collaborator/integrations/>
- Smartbear Software. (2021c). Retrieved 5th December 2021, from <https://smartbear.com/product/collaborator/features/artifact-review/>
- Taras, M. (2005). ASSESSMENT – SUMMATIVE AND FORMATIVE – SOME THEORETICAL REFLECTIONS. *British Journal of Educational Studies*, 53(4), 466–478. <https://doi.org/10.1111/j.1467-8527.2005.00307.x>

Appendices

A Initial e-mail to the lecturers

Laurvik (author):

”Hei, *navn*

Mitt navn er Torgeir Laurvik, jeg skriver masteroppgave for Hallvard Trætteberg. Masteroppgaven handler om hvordan ”kodereview” slik det gjøres i industrien kan inspirere et verktøy som kan brukes for å effektivisere (og forbedre) vurderingsprosessen i grunnleggende programmerings- og systemutviklingsemnene på NTNU.

Verktøyet jeg og Hallvard har i tankene skal altså ligne en del på hvordan kode presenteres ifm. merge request i GitLab, enten kommentarer som i Google docs eller merge request. Automatisk kjøring av tester hvis det er lagt inn av fagstab (ITGK og OOP) eller studentene (mest aktuelt for programvareutvikling og webutvikling). Unified diff-fil hvor studentenes kode får mest fokus, ikke kodeskjelett skrevet av fagstab. Statisk kodeanalyse for brudd på kodekonvensjoner, samarbeid om vurdering mellom flere sensorer (aktuelt ifm. lovendringen om to sensorer fra august av) etc. Vi har funnet ut at jeg ikke har tid til å realisere systemet, men jeg skal foreslå et system, slik at en evt. master/phd-student senere kan ta opp tråden.

Jeg ser at du står som emneansvarlig for *emnekode*.

I den forbindelse lurte jeg på hvordan vurderingen av eksamen gjøres i *emnekode* per nå. Lastes koden studentene skriver ned lokalt og kjøres? Inspiseres koden i Insperia (er det det som brukes i *emnekode*?) uten å kjøre den? Eller gjøres det på en annen måte?

Hva tenker du om dette? Er det noen funksjoner du tenker kan være nyttige for den bruken som er tenkt? Og som sagt, om jeg kunne få en forklaring på hvordan vurderingen utføres i dag i *emnekode*?”

B C++ email

The following conversation took place via e-mail, it is included based on permission from R. Sætre, the lecturer of the course and M. Bruland, the student which developed the HTML-based review supporting tool currently in use.

Sætre:

”Hei Torgeir,

Skulle gjerne hatt et slikt system du beskriver. Det er Martin som har laget den mest nyttige ressursen for oss, nemlig interaktive HTML-rapporter som viser studentkoden sammen men LF og utlevert kode. Kanskje han kan fortelle litt mer om systemet selv (hvis det ikke er fullt opp med eksamen nå?) Studentene våre leverte en zip-fil i Inspira etter at de hadde løst kodeoppgave i VScode. Siden vi har mange slike zip-filer og tilhørende karakterer ønsker vi å videreutvikle/trene og teste et system som kan komme frem til riktige slike karakterer automatisk... I praksis, før august.”

Bruland:

”Hei Torgeir,

Dette høres ut som et veldig spennende prosjekt som jeg tror absolutt kan være interessant. Når det gjelder hva vi kunne fått bruk for av funksjonalitet er jeg helt enig med Rune om at det hadde vært veldig bra med automatisert karaktersetting. Jeg skal i hvert fall prøve å gi en oversikt over systemet vi bruker, men hvis det er uklart kan jeg også finne tid til en liten prat på fredag eller neste uke. Studentene får lastet ned en zip fra Inspira som inneholder påbegynt kodeskjelett, og så løses oppgavene lokalt i kodeeditor. Per nå bruker vi et Docker-containerbasert enhetstestingssystem hvor besvarelsene (.zip-fil av mappestrukturen) parses og gjennomgår tester og potensielt får en poengsum per oppgave (men ikke karakter) basert på hvor stor andel av testene som blir bestått. Dette oppsummeres i en HTML-rapport per besvarelse, hvor besvarelse, kodeskjelett og løsningsforslag vises side om side for hver oppgave. Det er primært denne side-om-side visningen som har blitt brukt til sensur, mens poengsystemet ikke (så vidt jeg vet) har blitt testet og utviklet nok til å kunne foreslå karakter automatisk. Vi har tilgjengelig tester fra noen tidligere eksamener, så jeg vil tro at det for eksempel vil være mulig å teste endringer i systemet og sammenligne med faktisk sensur av besvarelser hvis det er relevant.

Testsystemet brukes til eksamen, mens de ukentlige øvingene leveres på Blackboard og godkjennes av studass på sal hvor studenten demonstrerer koden.”

C Web development email

The following conversation took place via e-mail, it is included based on permission from T. Aalberg, the lecturer of the course.

Laurvik:

”...I den forbindelse lurte jeg på hvordan vurderingen av eksamen gjøres i webdev per nå. Lastes koden studentene skriver ned lokalt og kjøres. Inspiseres koden i Insperia (er det det som brukes i webdev?) uten å kjøre den? Eller gjøres det på en annen måte?”

Aalberg:

”webapplikasjonene blir deployet på en vm så de er kjappe å prøve ut, kode inspiseres på GitLab, eller repoet klonet og kjøres lokalt.”

Laurvik:

”Jeg tok IT2810 for noen år siden, og da husker jeg øvingene også talte på karakteren. Siden det kun er senior-fagstaben som kan utføre sluttvurderingen i et fag, hvordan løses det med disse øvingene som det jo er et så stort volum av (regner med at det er mye mer ressurskrevende å vurdere enn eksamen)?”

Aalberg:

”I praksis er det jeg som gjør alle vurderinger, men jeg baserer meg en god del på tilbakemeldingene de får og etterprøver det meste. Reviewen de gjør handler mye om å sjekke tekniske krav og i mindre grad om kvalitative vurderinger på kodenivå (men handler litt om det også)”

Laurvik:

”Hva tenker du om dette verktøyet og nyttiligheten av det? Er det noen funksjoner du tenker kan være nyttige for den bruken som er tenkt? Og som sagt, om jeg kunne få en forklaring på hvordan vurderingen utføres i dag i webdev?”

Aalberg:

”Jo, har jo snakket med Hallvard om det og er helt enig i behovet. Prøvde å foreslå noe i dene gaten for egne masterstudenter, men ingen som tok oppgaven. Det jeg ville likt å se var et kodereview-system som tar hensyn til utdanningskonteksten: i praksis ting som automatisk fordeling av review, anonymisering, at faglærer kan inspiserer og følge med. Når studenter vurderer hverandre trenger vi også features som flagging og kommentarer tilbake til reviews de mener er feil, student-reviews er bare pålitelige hvis det er 4+ vurderinger som gjøres etc.

Jeg er vel egentlig mest interessert i features som gjør at en kan motivere til «student-aktive vurderingsformer». Når det gjelder webutvikling så er planen å gå over til bestått/ikke-bestått fordi den typen vurderinger vi gjør ikke vil la seg gjennomføre med eksterne sensorer. Ved bruk av bestått/ikke-bestått er det jo også andre muligheter for å lage seg et vurderingssystem hvor en benytter studentenes vurderinger av hverandre i faglærers endelige vurdering.”

D TDT4110 exercises description

Taken from the course' page at Blackboard LMS which is only accessible for students and employees at NTNU.

Øvinger

- Totalt 10 øvinger, hvorav 2 er auditorieøvinger
- Minst 8 av 10 øvinger, hvorav minst én auditorieøving må gjøres og bli godkjent for å kunne ta eksamen i emnet
- Øvingene er todelt, hvor begge deler må gjøres:
 - 1. Teori
 - 2. Programmering
- Oppgavene og informasjon om hvordan de gjøres og leveres inn ligger på BlackBoard
- Øvingene kan godt gjøres i grupper, men godkjennes individuelt.
- Hvis du har øvingsopplegget godkjent i et tidligere år vil det fortsatt telle som godkjent øvingsopplegg i år.
 - Dette gjelder også på tvers av Python-fagene TDT4109 og TDT4110 (og det gamle TDT4105). Det vil si at hvis du har godkjent øvingsopplegg i ett av fagene, slipper du å gjøre øvinger i det andre.
 - Det gjelder derimot *ikke* for Numerikk, TDT4127. Har du godkjent øvingsopplegg fra TDT4109 eller TDT4110 må du fremdeles gjøre øvingene i TDT4127, og motsatt.
 - Dersom du ønsker å ta eksamen på nytt må du huske å opprette vurderingsmelding i studentweb, dette skjer ikke automatisk.

Godkjenning / veiledning av øvinger

- Øvinger godkjennes på følgende måte:
 - 1. Lever følgende på Blackboard:
 - (i) Teori (50% korrekt kreves for godkjenning)
 - (ii) Programmeringsoppgaver
 - 2. Demonstrer programmeringsoppgavene for din læringsassistent
 - Under godkjenning må det redegjøres for hva som er gjort. Man må også kunne svare på spørsmål knyttet til øvingen

E TDT4110, exam 2020

Eksamen2 TDT4110

10/01/2020, 11:25

Hva står CPU for?

- Central processing unit
- Circuit processing unit
- Control programming unit

Hva er cache?

- Forgjengeren til dagens internett.
- En metode for å printe ledninger på chiper i flere lag.
- En veldig rask minneteknologi.

Hvilke 5 steg er med i "Fetch/Execute Cycle"?

- Instruction Fetch(IF), Instruction Execute(EX), Instruction Decode(ID), Data Decode(DD), Result Return(RR)
- Instruction Fetch(IF), Data Fetch(DF), Instruction Decode(ID), Result Return(RR), Instruction Execute(EX)
- Instruction Fetch(IF), Data Fetch(DF), Instruction Decode(ID), Data Decode(DD), Result Return(RR)

Hva er pipelining?

- En teknikk der en CPU kan utføre flere instruksjoner parallelt.
- En teknikk som fungerer som en sikker tunnel mellom din maskin og en tjener.
- En teknikk der man sender data mellom de forskjellige delene i maskinen i «pipes».

Hvilken av disse lagringsenhetene er IKKE en sekundærlagringsenhet?

- Hurtigbufferet i datamaskinen.
- En minnepinne
- En SSD satt rett i PCI Expressbussen

Hvilket heksadesimale tall representerer $(100111)_2$?

- 27
- 43
- 39

Navnet "Bob" skrives som "0100 0010 0110 1111 0110 0010" i Extended ASCII. Hvilket alternativ representerer ordet "obo" i Extended ASCII?

- 0110 1111 0100 0010 0110 1111
- 0110 1111 0110 0010 0110 1111
- 0110 0010 0100 0010 0110 0010

Informasjon som beskriver informasjon er kalt:

- Collating data
- Metadata
- Special data

Et bilde har 800*600 piksler, i 16 bit fargeformat. Hvor mye plass trenger det ukomprimert?

- Omtrent 1 MB
- Omtrent 2 MB
- Omtrent 500 kB

Hvilken av følgende komprimeringer er loss-less?

- Run-length coding
- MP3
- JPEG

Hva er phishing?

- At uvedkommende tar kontroll over en brukers datamaskin.
- Å opptre som en kjent nettside (f.eks. nettbank) for å få tak i personlig informasjon som f.eks. aksesskoder, kontonummer, etc.
- Å fiske etter personlig informasjon ved å late som om maskinen er under virusangrep.
- Angriperen lover brukeren at feilen skal rettes opp dersom brukeren først oppgir kontonummeret sitt.

Hva er scams?

- At uvedkommende tar kontroll over en brukers datamaskin.
- Bevisst blokkering av tilgang til en nettside eller tjeneste
- Å lure brukere av f.eks. en nettside eller en tjeneste til å investere penger eller gjøre noe ulovlig.

Hvordan fungerer replay-angrep?

- Pakker fra tidligere sesjoner fanges opp og sendes. For eksempel passord-pakker fra tidligere pålogginger.
- Bruker blir oppringt fra en tjeneste som ber brukeren ringe tilbake. Hvis brukeren ringer tilbake blir hun eller han belastet for store beløp for en ringetjeneste som koster mye å bruke.
- Chat-bot som svarer brukere på nettsider feilaktig for å innhente sensitiv informasjon.

Hva skjer når en melding krypteres?

- Dataene i meldingen endres, slik at kun riktig mottaker kan rekonstruere den opprinnelige meldingen.
- Dataene i meldingen deles i små pakker slik at mottaker er den eneste som får tak i dem alle.
- Meldingen må gå gjennom en brannmur, slik at det blir vanskeligere for angripere å få tak i dataene.

Hva er riktig om Payload encryption?

- Krypterer både pakkehodet og meldingsinnholdet i pakken.
- Krypterer kun meldingsinnholdet i pakken og ikke selve pakkehodet.
- Krypterer kun pakkehodet og ikke selve meldingsinnholdet i pakken.

Hva er sant om transportlaget?

- Transportlaget inneholder alle spesifikasjoner relatert til radiofrekvenser.
- Transportlaget består blant annet av spesifikasjoner om nettverksadressering og det maksimale antallet pakker som et nettverk kan støtte.
- Transportlaget sørger for at all data blir levert slik den ble sendt; komplett og i riktig rekkefølge.

Hva står forkortelsen WAN for i forbindelse med nettverk?

- World Area Network
- Wired Area Network
- Wide Area Network

Hva vil universal service si i forbindelse med nettverk?

- Å tillate kommunikasjon mellom datamaskiner uavhengig av hvilken type nettverk de sitter på.
- Å tilby streaming-tjenester for alle koplet til internett.
- Å tilby tilgang til skytjenester for alle datamaskiner.

Hva er masken til IPv4-adressen 255.255.128.0 i CIDR-notasjon?

- /36
- /256
- /17

Hva står TCP for i nettverkssammenheng?

- Transmission Control Protocol
- Transmission Channel Protocol
- Transport Channel Protocol

Maks poeng: 20

i Kodeforståelse (30%)

Velg det svaret du mener er mest riktig av alternativene. For hvert spørsmål gis det poeng på følgende måte:

- Korrekt avkrysning 3 poeng.
- Feil avkrysning 0 poeng
- Ingen avkrysning 0 poeng

2 Oppgave 2a

Hva skrives ut fra følgende kode?

```
def myst3(liste, navn):  
    dikt = {}  
    for i in liste:  
        dikt[i[0]] = f'{{sum(i[1:])/len(i)-1}}:{{.2f}}'  
    return navn, dikt  
  
liste = [['Per', 98, 67], ['Anne', 80, 66], ['Svein', 99, 88]]  
print(f'{{myst3(liste, "Anne")}}')
```

Velg ett alternativ

- ('Anne', {'Anne': '73.00'})
- ('Anne', {'Per': '82.50', 'Anne': '73.00', 'Svein': '93.50'})
- {'Per': 82.5, 'Anne': 73.0, 'Svein': 93.5}
- 'Anne' {'Per': '82,50', 'Anne': '73,00', 'Svein': '93,50'}
- ('Anne', {'Per': 82.5, 'Anne': 73.0, 'Svein': 93.5})
- {'Anne': '73.00'}

Maks poeng: 3

3 Oppgave 2b

Hva skrives ut fra denne kodebiten?

```
def myst(a):  
    b = []  
    while len(a) > 0:  
        c = a[0]  
        for i in a:  
            if i < c:  
                c = i  
        a.remove(c)  
        b.append(c)  
    return b  
  
print(myst([99, 17, 65, 19, 82]))
```

Velg ett alternativ

- [82,65,19,17,99]
- [17,82,65,19,99]
- [99,19,65,82,17]
- [65,19,17,99,82]
- [17,99,82,19,65]
- [17,19,65,82,99]

Maks poeng: 3

4 Oppgave 2c

Hva skrives ut fra denne koden?

```
def myst(a,b):  
    while b != 0:  
        c = b  
        b = a % b  
        a = c  
    return a  
  
print(myst(30,4))
```

Velg ett alternativ

- 3
- 4
- 1
- 2
- 5
- 6

Maks poeng: 3

5 Oppgave 2d

Hva skrives ut fra denne koden?

```
def myst2(a,b):  
    while len(a)%b != 0:  
        a = '0' + a  
    return a  
  
def myst(a,b,c):  
    d = ''  
    while a > 0:  
        d = str(a%b) + d  
        a = a//b  
    return myst2(d,c)  
  
print(myst(18,2,4))
```

Velg ett alternativ

- 00001001
- 01000010
- 00110010
- 00010010
- 0010010
- 100010

Maks poeng: 3

6 Oppgave 2e

Hva skrives ut av denne koden?

```
def myst(mat):  
    r = len(mat)  
    c = len(mat[0])  
    for i in range(r):  
        for j in range(c):  
            if mat[i][j] >= 6:  
                mat[i][j] = 0  
            else:  
                mat[i][j] = 1  
    return sum(mat[1])  
  
print(myst([[3, 2, 0], [19, 0, 18], [0, 6, 0]]))
```

Velg ett alternativ

- 1
- 6
- 2
- 48
- 37
- 5

Maks poeng: 3

7 Oppgave 2f

Hva skrives ut av denne koden?

```
def myst1(x,y,z):  
    t = x  
    x = y  
    y = t  
    for i in range(z):  
        z = x + y  
        y*= 2  
    return z  
  
print(myst1(2,4,6))
```

Velg ett alternativ

- 32
- 100
- 69
- 68
- 132
- 36

Maks poeng: 3

8 Oppgave 2g

Hva skrives ut av denne koden?

```
def myst(a,b,c):  
    d = ''  
    while a > 0:  
        d = str(a%b) + d  
        a = a//b  
    return d  
  
print(myst(18,2,4))
```

Velg ett alternativ

- 01110
- 11001
- 11100
- 10010
- 0010
- 1100

Maks poeng: 3

9 Oppgave 2h

Hva må *a* og *b* være for at programmet skal skrive ut MARTIN?

```
def myst(tekst,a,b):  
    return tekst[a::b].upper()  
  
print(myst('mgfmiyaieabarbnramisdtnaooeiehnnrnza',a,b))
```

Velg ett alternativ

- a = 3, b = 5
- a = 3, b = 6
- a = 3, b = 7
- a = 0, b = 6
- a = 1, b = 5
- a = 1, b = 6

Maks poeng: 3

10 Oppgave 2i

Hva skrives ut av denne koden?

```
def myst(a,b):  
    b = [[a[0],1]]  
    for i in range(1,len(a)):  
        funnet = False  
        for j in b:  
            if a[i] == j[0]:  
                j[1] += 1  
                funnet = True  
        if not funnet:  
            b.append([a[i],1])  
    return b  
  
tekst = 'DettE er en enkel streng. Eller to'  
b = []  
b = myst(tekst,b)  
print(b[1])
```

Velg ett alternativ

- IndexError: list index out of range
- ['e',8]
- ['e',7]
- ['e','7']
- ['e',9]
- ['e']

Maks poeng: 3

11 Oppgave 2j

Hva skrives ut av denne koden?

```
import numpy as np  
  
a = np.arange(1,5,0.5)  
c = a.size  
b = (a+c)*2  
print(b)
```

Velg ett alternativ

- [18. 19. 20. 21. 22. 23. 24.]
- [18 19 20 21 22 23 24 25]
- [18. 19. 20. 21. 22. 23. 24. 25.]
- [18., 19., 20., 21., 22., 23., 24., 25.]
- [18., 19., 20., 21., 22., 23., 24., 25.,26.]
- [18. 19. 20. 21. 22. 23. 24. 25. 26.]

Maks poeng: 3

```
# module nameregister

'''
days_in_month(month)
The function takes an int between 1 and 12, and returns the number
of days in that month. Leap years are ignored.
The function crashes for numbers outside the range 1-12.
Example: days_in_month(1) returns 31
'''
def days_in_month(month):
    return [31,28,31,30,31,30,31,31,30,31,30,31][month-1]

'''
register_name(people, fnr, name)
Used to add persons to the dictionary 'people'
The function takes three parameters:
people: Dictionary containing personal ID number (key) and name (value)
fnr (string): Personal ID number (fødselsnummer) of the person to be added
name (string): Name of the person to be added

You're not to write this - only use it
'''
def register_name(people, fnr, name):
    people[fnr] = name

'''
find_name(people, fnr)
Used to find the name of a person in 'people', if you have the ID number.
The function takes to parameters:
people: Dictionary containing personal ID number (key) and name (value)
fnr (string): Personal ID number of the person to look up
The function returns the name of the person if it exists. If not, it
returns an empty string, ''

You're not to write this - only use it
'''
def find_name(people, fnr):
    return people.get(fnr, '')
```

i Programmering (50%)

Førerkortprikker og botbetaling

I denne eksamenen skal du lage et system for registrering av prikker som norske borgere får når de begår trafikkforseelser. Du skal lage et sett funksjoner, som oppfyller en del krav til systemet, disse blir beskrevet under hver oppgave. Dette er et oversiktsbilde av systemet, så følger en beskrivelse av datastrukturen som skal lages og modulen som allerede eksisterer. Merk for øvrig at dette oppgavesettet kun er inspirert av det faktisk prikkbelastningssystemet som finnes i Norge.

Modulen *nameregister.py*

Alle personer som registreres i systemet må registreres med fødselsnummer og navn (fornavn etternavn). Begge disse er av type strenger. Modulen *nameregister* inneholder tre funksjoner som hjelper deg med å registrere denne koblingen. Modulen ligger i den samme folderen som koden du skal skrive, du skal altså bruke disse i koden din. De tre funksjonene

12 Oppgave 3a - Dato sjekk (7%)

Nye førere må registreres (mer om det senere), og da må en legge inn folks fødselsnummer (streng med 11 siffer). Vi ønsker ikke å legge inn verdier som er feil i systemet. Derfor må det kontrolleres at datoer som legges inn i alle fall er reelle.

Skriv funksjonen **check_date**

- Funksjonen skal ta inn en streng som skal verifiseres som en reell dato. Strengen er hele fødselsnummeret til føreren, der de seks første dekker fødselsdatoen. Disse er på formatet ddmmyy - 12 mars 1979 blir da '120379'. Litt informasjon og krav:
- Alder: år personen er født (kun de to siste sifrene). Du kan se bort fra alder på personene, altså ingen sjekk om på om de gamle nok til å ha førerkort.
- Måned: tallet må være mellom 01 og 12, og alltid med to siffer (mars blir '03')
- Dag: dag passer inn med måneden brukt.
- Det finnes allerede en funksjon `days_in_month(month)`. Denne ligger i modulen `nameregister.py` i den samme mappen som programmet ditt. Denne må du gjerne bruke, og du finner en beskrivelse av den nederst på funksjonsarket.
- Du kan også se bort fra alt som har med skuddår å gjøre.
- Hvis datoen er reell skal funksjonen returnere `True`, ellers skal den returnere `False`.

Eksempel:

```
>>> print(check_date('29127311245'))
True
>>> print(check_date('31117532456'))
False
```

Skriv ditt svar her...

1	
---	--

Maks poeng: 7

13 Oppgave 3b - Registrer ny fører (8%)

Når en ny person begår en trafikkforseelse må denne registreres. Her er det to ulike systemer dette skal registreres i - *people* og *register*.

Skriv funksjonen **register_person**

Funksjonen tar fire parametre:

- *people*: En dictionary som gir kobling mellom fødselsnummer og navn.
- *register*: En dictionary som har fødselsnummer (streng) som nøkkel og en liste over prikker som verdi.
- *fnr*: En streng som inneholder fødselsnummeret til den som skal registreres. Du kan forvente at datoen her er korrekt, og at det ikke finnes noen med dette fødselsnummeret fra før.
- *name*: En streng som inneholder navnet til den som skal registreres.

Funksjonen skal gjøre følgende:

- I *people* skal du legge til et nytt innslag med *fnr* og navn. *register_name* i modulen *nameregister* bør være til hjelp.

- I *register* skal det registreres et nytt innslag med nøkkel *fnr*, verdien skal være en tom liste.
- Variablene *people* og *register* kan du forutsette at du har tilgang på allerede - du trenger altså ikke lage dem. Dette gjelder også seinere oppgaver.

Eksempel:

```
>>> print(people) # Empty dictionary
{}
>>> print(register) # Empty dictionary
{}
>>> register_person(register, people, '11010154321', 'Terje Rydland')
>>> register_person(register, people, '23056644521', 'Børge Haugset')
>>> print(people)
{'11010154321': 'Terje Rydland', '23056644521': 'Børge Haugset'}
>>> print(register)
{'11010154321': [], '23056644521': []}
```

Skriv ditt svar her...

1	
---	--

Maks poeng: 8

14 Oppgave 3c - Telle prikker (7%)

Personer får registrert prikker i en dictionary kalt *register*. Denne har som nøkkel et personnummer (streng med 11 siffer). Verdien er en liste som inneholder alle prikkene

personen har fått, den første først.

Skriv funksjonen **count_dots**

- Du kan forvente at du har tilgang til variabelen *register*
- Funksjonen skal ha to parametre:
 - Den første er *register* (dictionary), som er nevnt over.
 - Den andre er *fnr* (streng, 11 siffer), fødselsnummeret til føreren man skal finne antall prikker til.
- Funksjonen skal returnere summen av prikker (som heltall) denne personen har fått.
- Du kan forvente at fødselsnummeret som oppgis finnes i *register*.

Eksempel:

```
>>> # They have respectively 8 and 3 dots
>>> register = {'11010154321':[3,2,3], '23056644521':[3]}
>>> count_dots(register, '11010154321')
8
```

Skriv ditt svar her...

1	
---	--

Maks poeng: 7

15 Oppgave 3d - Legg til prikker (7%)

Prikker legges inn som verdi i dictionary *register*, knyttet til nøkkelen fødselsnummer (streng med 11 siffer). Prikkene er lagret i listeform, og nye prikker registreres på slutten av den eksisterende listen. Listen består av heltall, og kan se slik ut: [2, 2, 3]. Når nye prikker har blitt lagt til må en sjekke om totalt antall prikker er 10 eller mer. Hvis de er det skal det skrives ut en beskjed om at føreren har gått over grensen, og førerkortet skal beslaglegges. Her holder det å skrive ut en beskjed til brukeren som legger inn prikkene.

Skriv funksjonen **add_dots**

- Funksjonen skal ha tre parametre
 - *register* - dictionary som inneholder fødselsnummer og prikker for alle, som før.
 - *fnr* - fødselsnummeret til den som har kjørt i grøfta. Du kan forvente at dette fødselsnummeret allerede er registrert.
 - *dots* - antallet prikker som skal legges til i slutten av denne førerens liste.
- Hvis antallet prikker for føreren etter innlegging av nye prikker er 10 eller mer, skal det skrives ut en beskjed. Se eksempelet under. Husk funksjoner som er laget før selv om du ikke har skrevet dem selv.
- Du kan forvente å ha tilgang til variablene *people* og *register*.

Eksempel:

```
>>> print(register)
{'11010154321': [3, 2, 3], '23056644521': [3]}
>>> add_dots(register, '11010154321', 3)
The driver has more than 10 dots, confiscate!
>>> add_dots(register, '23056644521', 3) # Børge previously had 3
>>> # Nothing is printed, as the total is less than ten.
```

Skriv ditt svar her...

1	
---	--

Maks poeng: 7

16 Oppgave 3e - Manuell innlegging av prikker (7%)

Funksjonene som er skrevet til nå har alle basert seg på at informasjon som skal legges inn allerede er definert. Nå har vi kommet til delen der brukeren av systemet må skrive inn ting. Skriv funksjonen **manual_registration**

- Funksjonen har to parametre - *register* og *people*. De er like som i oppgavene før.
- Brukeren skal først skrive inn et fødselsnummer (streng med 11 siffer).
- Det må sjekkes om fødselsdatoen i fødselsnummeret er en reell dato. Hvis ikke - avbryt funksjonen. Fødselsdatoen er de seks første sifrene i fødselsnummeret.
- Hvis dette fødselsnummeret ikke allerede er registrert må denne nye personen registreres i både *people* og *register*, i tråd med funksjonsbeskrivelsene gitt i tidligere oppgaver. Husk å spørre om navn!
- Til slutt skal funksjonen spørre om hvor mange prikker denne føreren har fått, og registrere dem.

Eksempel:

```
>>> # Existing user:
>>> manual_registration(register, people)
Personal ID number: 23056644521
Dots to register: 2
>>> # Invalid date of birth:
>>> manual_registration(register, people)
Personal ID number: 12345678909
The date is invalid.
>>> # New ID number
>>> manual_registration(register, people)
Personal ID number: 30118866154
Not registered before. Name: Alf Inge Wang
Dots to register: 1
```

Skriv ditt svar her...

1	
---	--

Maks poeng: 7

17 Oppgave 3f - Betale seg ut av problemer (7%)

Det er alltid mulig å betale seg ut av problemer i dette systemet. En fører har mulighet til å betale for å redusere antallet prikker som er registrert på seg.

Skriv funksjonen **pay**

Funksjonen **pay** har en parameter:

- *register*: Registeret over fødselsnummer og nåværende prikker.

Den fungerer på følgende måte:

- Brukeren skal bli spurt om å skrive inn fødselsnummeret og summen føreren skal betale.
- Funksjonen skal legge inn negative prikker. 1 prikk per 10 000 kroner. Hvis en fører vil betale 35 000 kroner skal tallet -3 legges til på slutten av listen, som i eksempelet under. Husk tidligere funksjoner.
- Hvis føreren ikke er registrert skal meldingen 'Fører ikke registrert' skrives ut og funksjonen avsluttes.

Eksempel:

```
>>> register = {'11010154321': [3, 2, 3, 2], '23056644521': [3]}
>>> pay(register)
Personal ID number: 11010154321
How much will the driver pay? 35000
>>> print(register['11010154321'])
[3, 2, 3, 2, -3]
```

Skriv ditt svar her...

1	
---	--

Maks poeng: 7

18 Oppgave 3g - Lagring til fil (7%)

Vi ønsker å lagre informasjon om førerne som er registrert på fil.

Skriv funksjonen **save_to_file**

- Funksjonen tar to parametre, *register* og *people*
- Filen det skal skrives til skal hete *dots.txt*.
- For hver av de registrerte førerne skal det lages en linje der følgende streng står (uten ' foran og bak): '11010154321 (Terje Rydland) har 7 prikker.'

Eksempel:

```
>>> register = {'11010154321': [3, 2, 3, 2], '23056644521': [3]}
>>> people = {'11010154321': 'Terje Rydland', '23056644521': 'Børge Haugset'}
>>> save_to_file(register, people)
```

Etter at koden over har kjørt, vil *dots.txt* inneholde følgende:

11010154321 (Terje Rydland) har 10 prikker.

23056644521 (Børge Haugset) har 3 prikker.

F TDT4100, exam 2017

If you feel necessary information is missing, state the assumptions you find it necessary to make. If you are not able to *implement* classes and method that a part asks for, you may still *use* these classes and methods later.

An overview of classes and methods for all the parts are provided in appendix 1. The comments contain requirements for the various programming tasks, that must be considered when you solve them. Feel free to define extra methods, in addition to those provided, to make your solution tidier. Useful standard classes and methods can be found in appendix 2.

The topic is a diner (**Diner**) and the problem is seating (**Seating**) groups (**Group**) of guests at the tables (**Table**).

Part 1 – The Group, Table and Seating classes (15%)

The **Group**, **Table** and **Seating** classes (appendix 1) are so-called value classes, with data that must be provided when objects are created and cannot be changed later. **Group** must contain data about the number of guests in the group, **Table** must contain data about the number of seats (capacity) and **Seating** must keep track of the table a group is seated at.

- Finish the **Group** and **Seating** classes, including necessary encapsulation methods.
- It should not be possible to have **Seating** objects for tables without enough seats for the whole group seated there. Write the code needed for enforcing this rule.
- Assume **Group** had a method for changing the number of guests. Explain with text and/or code what changes you would need to enforce the rule in b).
- In addition to the number of seats, a table must have a table number. This number must be a unique counter that is not provided, but is automatically set by code in the **Table** class itself when **Table** objects are created. The very first table that is created should have number 1, the second have number 2, and so forth. Implement the constructor and other necessary code, including the **getNum** method!

Part 2 – The Diner class (40%)

The **Diner** class (appendix 1) keeps track of tables and seatings, i.e. which groups are seated at which tables.

- Write the necessary field declarations and constructor(s), given that the diner has more than one table. Also write the methods for adding and removing tables.
- Write the **isOccupied** and **getCapacity** methods.
- Tables can be merged, typically to make room for large groups of guests. Tables can correspondingly be split, to avoid that a small group occupies a large table. Write the **mergeTables** and **splitTable** methods. At this point, you don't need to represent which tables are actually merged, they just disappear, and must be re-created when split.
- Draw an object state diagram that illustrates the behavior of **mergeTables**.

- e) When guests are seated you must find the smallest, available table with enough capacity. Write the **hasCapacity** and **findAvailableTables** methods. Also write other code necessary for ensuring return value of **findAvailableTables** is sorted.
- f) A new seating of guests is registered in a **Seating** object. Write the **createSeating**, **addSeating** and **removeSeating** methods.

Part 3 – The Table, SimpleTable and CompositeTable classes (15%)

A problem when merging and splitting tables is that the table numbering becomes wrong, when the logically same table is re-created and is assigned a different number. One way of handling this is to have two table types, simple tables (**SimpleTable**) and composite tables (**CompositeTable**), where the latter keeps track of the tables that are merged. This requires a new version of the **mergeTable** method that must create a **CompositeTable** containing the two tables that are merged, and the **splitTable** method must be re-written so it splits the a **CompositeTable** into the *same* two tables that were merged. The **splitTable** method does not need the two **capacity** arguments any more because the tables know their capacity.

- a) Explain with text and/or code how you will use inheritance and/or interfaces, so **Table** still can be used as a general table type and **SimpleTable** and **CompositeTable** can handle respective special cases. Also explain the behavior of **SimpleTable** and **CompositeTable**.
- b) Write new versions of **Diner**'s **mergeTable** and **splitTable** methods. Note that the new **splitTable** method only takes a **CompositeTable** argument.

Part 4 – The GuestManager class (20%)

Guests arriving at a **Diner** are received by a corresponding **GuestManager** (see appendix 1), that tries to seat them. If it fails, the guests must wait for a table with enough seats to become available. Hence, **GuestManager** needs to track how the capacity of the **Diner** object changes. This is done by making **Diner**'s capacity property, as returned by a call to **getCapacity(false)**, *observable*.

- a) What does observability entail? Briefly explain with text and/or code how to make a (property of a) class observable.
- b) Explain with text and/or code how you would modify **Diner** so **GuestManager** can listen to changes to the capacity property (by implementing the **CapacityListener** interface).
- c) Explain with text and/or code how you would write the **GuestManager** class. We don't expect automatically merging and splitting of tables, but those arriving first should preferably be seated first.

Part 5 – Misc. (10%)

- a) Is **CapacityListener** a *functional* interface? Explain your answer!
- b) (Re)write one of **isOccupied** or **getCapacity** in **Diner** so it uses the **Stream** technique and the function syntax of Java 8 (if you haven't already, that is!).
- c) Explain with text and/or code how you would test the **isOccupied** method in **Diner** in a separate **DinerTest** class, and mention which methods in **Diner** you would use and how **Diner** if necessary must be modified for **isOccupied** to be easily testable.

Appendix 1: Provided code (fragments)

```
// part 1

/**
 * A group (of people) dining together, and should be seated at the same table.
 * We currently only need to handle the size.
 */
public class Group {

    /**
     * Initializes this Group with the provided guest count
     */
    public Group(int guestCount) {
        ...
    }
}

/**
 * A table with a certain maximum capacity.
 */
public class Table {

    /**
     * Initializes this Table with the provided capacity.
     * The table is also assigned a unique number.
     * @param capacity
     */
    public Table(int capacity) {
        ...
    }

    /**
     * @return the table number
     */
    public int getNum() {
        ...
    }
}

/**
 * Represents the fact that a Group is seated at and occupies a Table
 */
public class Seating {

    /**
     * Initializes this Seating ...
     */
    public Seating(...) {
        ...
    }
}
```



```

// part 2

/**
 * A place where groups of guests can buy a meal
 */
public class Diner {

    /**
     * Tells whether a Table is occupied.
     * @param table the Table to check
     * @return true if anyone is sitting at the provided Table
     */
    public boolean isOccupied(Table table) {
        ...
    }

    /**
     * Computes the guest capacity,
     * either the remaining (includeOccupied == false) or total (includeOccupied == true).
     * @param includeOccupied controls whether to include tables that are occupied.
     * @return the guest capacity
     */
    public int getCapacity(boolean includeOccupied) {
        ...
    }

    /**
     * Adds a table to this Diner
     * @param table
     */
    public void addTable(Table table) {
        ...
    }

    /**
     * Removes a Table from this Diner.
     * If the table is occupied an IllegalArgumentException exception should be thrown.
     * @param table
     * @throws IllegalArgumentException
     */
    public void removeTable(Table table) {
        ...
    }

    /**
     * Merges two tables, i.e. replaces two tables with one table.
     * lostCapacity is the difference between the old capacity and the new.
     * This number is typically positive, since seats are lost when moving two tables
     * close to each other.
     * @param table1
     * @param table2
     * @param lostCapacity
     * @throws IllegalArgumentException if any of the tables are occupied
     */
    public void mergeTables(Table table1, Table table2, int lostCapacity) {
        ...
    }

    /**
     * Splits a table into two, i.e. replaces one tables with two tables.
     * The two capacities are the capacities of the two new tables.
     * @param table
     * @param capacity1
     * @param capacity2

```

```

    * @throws IllegalArgumentException if the table is occupied
    */
    public void splitTable(Table table, int capacity1, int capacity2) {
        ...
    }

    /**
     * Tells whether a table has the provided capacity,
     * i.e. if that number of new guests can be seated there.
     * Note that a table cannot be shared among different groups.
     * @param table
     * @param capacity
     * @return true if capacity number of guests can be seated here, false otherwise.
     */
    public boolean hasCapacity(Table table, int capacity) {
        ...
    }

    /**
     * Returns the tables that has the provided capacity.
     * The tables should be sorted with the one with the least capacity (but enough) first.
     * @param capacity
     * @return the tables that has the provided capacity
     */
    public Collection<Table> findAvailableTables(int capacity) {
        ...
    }

    /**
     * Finds a suitable, existing table for the provided group, and creates
     * (but doesn't add) a corresponding Seating.
     * The chosen table should be the one with the least capacity.
     * @param group the group to be seated
     * @return the newly created Seating
     */
    public Seating createSeating(Group group) {
        ...
    }

    /**
     * Creates and adds a Seating for the provided group, using the createSeating method.
     * @param group
     * @return true if a Seating was created and added, false otherwise.
     */
    public boolean addSeating(Group group) {
        ...
    }

    /**
     * Removes the seating for the provided table (number), if one exists
     * @param tableNum the number of the table to be removed
     */
    public void removeSeating(int tableNum) {
        ...
    }
}

```

```

// part 3
public class SimpleTable ... Table {
    public SimpleTable(int capacity) {
        ...
    }
    ...
}

/**
 * A table that consists of two other tables.
 */
public class CompositeTable ... Table {
    public CompositeTable(Table table1, Table table2, int lostCapacity) {
        ...
    }
    ...
}

// part 4

/**
 * Interface for listening to changes in Diner capacity
 */
public interface CapacityListener {
    /**
     * Called to inform that a Diner has changed capacity
     * @param diner
     */
    public void capacityChanged(Diner diner);
}

/**
 * Handles guests arriving at and departing from a Diner.
 */
public class GuestManager ... {
    public GuestManager(Diner diner) {
        ...
    }

    /**
     * Handles arriving groups, by either seating them immediately
     * (if possible) or putting them in queue. Those enqueued will
     * be seated when the Diner's (change in) capacity allows.
     * @param group
     */
    public void groupArrived(Group group) {
        ...
    }

    /**
     * Handles departing groups, by removing their seating.
     * @param tableNum the table where the group was seated
     */
    public void groupDeparted(int tableNum) {
        ...
    }
    ...
}

```

G TDT4102, exam 2019

Assignment 1: Reading code (20 %)

Answer as follows on a regular answer sheet:

1a) ... your answer here

1b) ... your answer here ... and so on

1a) What is printed?	<pre>int a = 20; float b = a * 2 + 1; int c = b++; b += b / 100; cout << a << " " << b << " " << c << endl;</pre>
----------------------	---

1b) What is printed?	<pre>int i = 1; int j = 1; while (j < 10) { j += i; i = j - i; cout << j << " "; } cout << endl;</pre>
----------------------	---

1c)	<pre>int f(char h, char *e, char &i) { h++; *e = h; i += 2; return h; }</pre>	<pre>char h = 'h'; char e = 'e'; char i = 'i'; h = f(h, &e, i); cout << h << e << i << endl;</pre>
Given the function f to the left, what is printed by the program to the right?		

1d)	<pre>int safe(int n, int d) { if (d == 0) { throw d; } return n / d; }</pre>	<pre>try { for (int n = 4; n > 0; n--) { cout << n; cout << safe(n, n - 2) << " "; } } catch (int e) { cout << "! "; } cout << endl;</pre>
Given the function safe to the left, what is printed by the program to the right?		

1e)	<pre>string apply(string s, map<char, string> &rules) { string s2 = ""; for (int i = 0; i < s.length(); i++) { s2 += rules[s[i]]; } return s2; }</pre>	<pre>map<char, string> rules; rules['A'] = "AB"; rules['B'] = "A"; string s = "A"; for (int i = 0; i < 4; i++) { cout << s << " "; s = apply(s, rules); } cout << endl;</pre>
Given the function apply to the left, what is printed by the code to the right?		

1f) What is printed?	<pre>string a = "trol"; string b = "rolo"; char c = 't'; for (int i = 0; i < 7; i++) { int k = a.find(c); cout << a[k]; c = b[k]; } cout << endl;</pre>
----------------------	---

1g)	<pre>char rot13(char c) { if (c >= 'a' && c <= 'z') { c -= 'a'; c = 'a' + ((c + 13) % 26); } return c; } string rot13(string s) { for (int i = 0; i < s.size(); i++) { s[i] = rot13(s[i]); } return s; }</pre>	<pre>string msg = "catz"; cout << rot13('a') << rot13('n') << rot13('z') << " "; cout << rot13(msg) << " "; cout << rot13(rot13(msg)) << endl;</pre>
-----	---	--

Given the function rot13 to the left, what is printed by the code to the right? (Here you need to know that the 26 lowercase letters in the English alphabet come after each other in the ASCII table, as «abcdefghijklmnopqrstuvwxyz».)

1h)	<pre>class S { public: S() {} char f() { return 'S'; } }; class U : public S { public: char f() { return 'U'; } };</pre>	<pre>S s; U u; S& r = u; S* p = &r; cout << s.f() << u.f() << r.f() << p->f() << endl;</pre>
-----	---	---

What is printed by the code to the right given the classes declared to the left?

Assignment 2: Linear regression (25 %)

Linear regression is an analytical technique which was employed as early as the beginning of the 1800s. The technique attempts to find a linear relationship between variables x and y on the form

$$y = ax + b$$

The starting point is a collection of data for x and y , and in simple linear regression we calculate the straight line which best describes the relationship between x and y . An example is shown in Figure 1. This is a statistical method and we will describe the mathematics you need to program solutions to the assignments. (Linear regression has many applications and is used together with other more advanced methods in e.g. "new" fields such as "data science" and artificial intelligence (AI).)

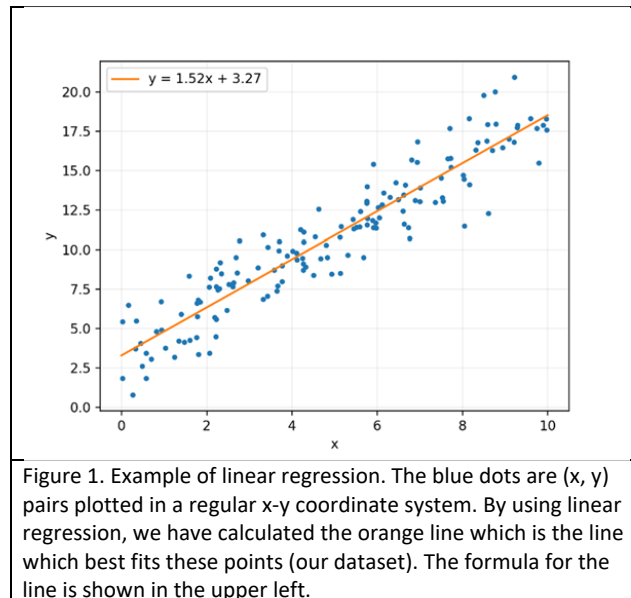


Figure 1. Example of linear regression. The blue dots are (x, y) pairs plotted in a regular x - y coordinate system. By using linear regression, we have calculated the orange line which is the line which best fits these points (our dataset). The formula for the line is shown in the upper left.

2a) Implement the function

`double sum(vector<double>& x)` which returns the sum of all the numbers in a vector x .

2b) Implement the function `double mean(vector<double>& x)` which returns the mean of the numbers in the vector x .

2c) Comma-separated values (CSV) is a common file format used to store data in table form (rows and columns). A CSV-file has one row per line, and each line is divided in columns separated by comma or space. An excerpt of a CSV-file used for this assignment is shown in

Figure 2. Implement the function `void read_csv(string filename, vector<double>& x, vector<double>& y)` which reads data from a CSV file with columns x and y . Here `filename` is the name of the CSV file and x and y are references to vectors of `double` which should be filled with data from the file, where the first column is x and the second column is y . You can assume that the columns in the CSV-file are separated by a space. If the file cannot be opened, the function should throw an exception as a single string consisting of «`Couldn't read file`» and the filename `filename`.

70	152
85	214
83	203
77	219
56	152
90	215

Figure 2.

2d) We are now going to implement code for linear regression. Given two vectors with data x and y , we wish to find a straight line which best describes the relationship between x and y . From mathematics we know that a straight line is given by the formula $y = ax + b$ where a is the slope and b is the point where the line crosses the y axis (i.e. when $x=0$). (For linear regression we are here using the "method of least squares" invented by Gauss: the line should be drawn through the given points such that the sum of the squared distances from the points to the line is minimized, where the distance is measured in the y direction.) Below are the formulas you should use for this case:

$var = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - \bar{x})^2$	$cov = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - \bar{x})(y_i - \bar{y})$
$a = \frac{cov}{var}$	$b = \bar{y} - a\bar{x}$

Here Σ means sum over all indices $i = 0 \dots n - 1$, where x_i is an element in the vector x . Furthermore, \bar{x} is common notation for the mean value of x . The

same for y . Note that you do not need to understand the mathematics behind the formulas, and you can solve the later sub-assignments without solving this particular sub-assignment.

Implement the function `pair<double, double> linreg(vector<double>& x, vector<double>& y)` which calculates values for a and b as described above. The function should return both values as a `std::pair p` where `p.first` is a and `p.second` is b .

2e) Now that we have found a line which best fits our dataset, we can use the line to predict a value for y given an arbitrary x value. Implement the function `vector<double> linpred(vector<double>& x, double a, double b)` which returns a vector with all y values given by using $y = ax + b$ for each of the x values in the vector x .

2f) Although we have found a straight line, it is not necessarily a good fit. A measure for how well the line fits our dataset is R^2 which is a number between 0 and 1 where 0 means worst and 1 means perfect. R^2 is calculated using the formula:

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1} (y_i - y_{\text{pred}_i})^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2}$$

Where y_{pred_i} is element no i from the vector returned by the function `linpred()` in the previous sub-assignment. Implement the function `double r2(vector<double>& y, vector<double>& y_pred)` which compares y (the observed values) and y_{pred} (the predicted values, from `linpred`) by calculating R^2 given by the formula above.

2g) We are now ready to use linear regression to find a relationship between two vectors of data x and y . Write a main-function which:

1. Reads x - and y -data from the file "data.csv"
2. Computes a and b using linear regression
3. Calculates R^2 for the line $y = ax + b$
4. Prints to the console the values for a , b , og R^2

Assignment 3: City Bikes (25%)

NTNU is focused on both the environment and students' health and has, together with cycle-general Ricardo, placed city bikes at different stations in Trondheim. You are employed by the Rector to program an application *GunnarBikes* to help Ricardo know where the bikes are and to organize transport of bikes so that all stations have a suitable number of bikes every morning.

3a) The program uses a data type named `Location` shown in Figure 4, where `name` is a *unique* name for the location and `p` is its position on a map. The data type

```
struct Location {
    string name;
    Point p;
    Location(string str, Point pt);
};
```

Figure 4. The data type `Location`

`Point` is from the textbook graphics library (see appendix). **Implement the constructor** for `Location` (declared in Figure 4) *outside* the class declaration and use initializer list.



Figure 3. Part of screenshot from the assignment City Bikes.

3b) Define a class `BikeStation` which contains the following *private* member variables: `loc` of type `Location`, `capacity` and `bikes` which both should be *unsigned int*, and `display` which should be a `Vector_ref<Shape>` and is used for the graphics part of the assignment. Furthermore, the class should contain *declarations* for a set- and a get-function for the member variables `bikes`, and an inline implementation of a function `getName()` which should return `loc.name`. (The final version of this class will contain more, but here you should only focus on what is mentioned in this sub-assignment.)

3c) Implement the set- and get-functions for `bikes` which you declared in the previous sub-assignment.

3d) Write a member function `string BikeStation::status()` which should return a string containing a short textual report with the status of a bike station. The format of the string is shown in Figure 3 as "10 out of 30" where the first number is the value of `bikes` (how many bikes are in the station) and the other number is the value of `capacity` (number of parking spots for city bikes).

3e) Write the constructor for `BikeStation`

```
BikeStation::BikeStation(Location where, unsigned int cap, unsigned int numBikes)
```

The constructor should initialize `loc`, `capacity` and `bikes` with values from the parameter list. Furthermore, the constructor should add pointers to the following *graphical elements* into the member variable `display`:

- 1) a pointer to a `Rectangle` object whose upper left corner is the point given by the member variable `loc` (see sub-assignment 3b). The width and height are given by two constants `dispWidth` and `dispHeight` which are available from your code. The rectangle should be filled with the color `Color::white`.
- 2) a pointer to a `Text` object with the name of the station, positioned at the same point as the rectangle, with the color `Color::blue`. The text should have font-size 20.
- 3) a pointer to a `Text` object which shows the status of the station, by showing the string returned by `status()` from the previous sub-assignment in the white rectangle, positioned 2 pixels from the rectangle's left edge and 15 pixels below the rectangle's upper edge. The text should have the color `Color::black`, and you do not need to worry about the font-size (standard/default is fine).

See also Figure 3 and use the information from the appendix.

3f) To test the application *GunnarBikes*, we wish to simulate a day of cycling. You shall write a function `simulateOneDay()` which takes as argument a vector of pointers to all the city bike stations (`vector<BikeStation*>`). The function should do the following:

- Attempt to perform `ridesPerDay` number of bike trips, where `ridesPerDay` is a given constant.
- For each *potential* bike trip, a *random* station should be selected where one wishes to take out a bike. Use `rand()` from the standard library (see the appendix) to select a random station to take out a bike and a random station to park it. This represents a desired bike trip.
- To ensure that a cyclist is guaranteed that he or she can leave the bike at the destination station, it should be possible to reserve an available parking spot at the destination station when you take out a bike. This ensures that a bike trip can be performed. A sufficient and necessary condition for a successful bike ride is hence that there exists at least one available bike at the departure station and that there is at least one available parking spot at the destination station. For each desired bike ride (from the previous point), we need to check this condition, and would like to collect statistics on the number of unsuccessful bike rides. This number is counted in a `map<string, int>` which is returned by the function. Here the key is the unique name of the desired departure station, and the integer is the number of unsuccessful bike rides from that station.
- You can assume that `BikeStation` also has a get-function `unsigned int getCapacity()` to read the variable capacity.

3g) Write a function `printStats()` which prints to the console an overview of unsuccessful bike rides as shown in Figure 5. The function takes as input a map of the same type as the previous sub-assignment, and the lines should be alphabetically sorted by the name of the station ("Festningen" comes before "Marinen" etc.)

Unsuccessful rides: 70 bike trips refused at Festningen 101 bike trips refused at Marinen 180 bike trips refused at Pirbadet 62 bike trips refused at Samfundet

Figure 5.

Assignment 4: Ring buffer and testing (30%)

A buffer is an area in memory used to temporarily store data before it can be processed further. In a circular buffer, or a *ring buffer*, you return to the beginning when you write (or read) past the last element. Figure 6 shows one possible state of such a buffer with 5 positions, see also the table below. A ring buffer is often used when data is to be transferred between two different devices and processed by the receiver in the same order it is received (like a queue). Computer memory is linear, so in order for it to "look" as it is circular, we need some extra program logic.

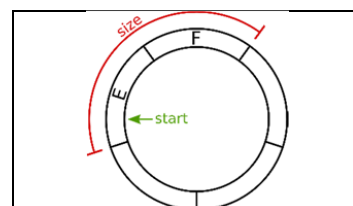


Figure 6. The ring buffer in state 4. The beginning of the buffer (position 0) is at the top of the figure and the buffer goes clockwise. "start" points at the first valid element in the buffer (here `start=4`), and "size" denotes the number of valid elements stored (here `size=2`).

In a ring buffer we need to keep track of where in the buffer valid data begins (*start*), as well as how much valid data is stored (*size*). In addition, we need to know the maximum capacity of the buffer (*capacity*). In the table below, we have shown 5 different states for a buffer with space for 5 elements (*capacity*=5).

State no.	Buffer contents	Value of start and size	Next operation on the ring buffer rb
1	[_ _ _ _ _]	start=0, size=0	rb.write("ABC");
2	[A B C _ _]	start=0, size=3	rb.write("DEF");
3	[F B C D E]	start=1, size=5	s = rb.read(3);
4	[F _ _ _ E]	start=4, size=2	s = rb.read(-1);
5	[_ _ _ _ _]	start=1, size=0	

In the beginning (state no. 1) the buffer is empty and *start* = 0 and *size* = 0. If we *write* three elements A B C to the buffer, the result is shown in state 2. The operation (code) to do this is shown in the last column. Note that the position in the buffer is numbered from 0 and up to *size* - 1. If we now write three new elements D E F to the buffer, we will write past the end of the buffer and return to the beginning, as shown in state 3. We will thus overwrite the first element. Since we overwrite the start of valid data, *start* is incremented to point at the next element B (which is stored in position 1). Notice that now *size* = *capacity* since the buffer is full.

Reading from the ring buffer will free space corresponding to the number of elements read. If we read 3 elements, the buffer returns B C D. After the read, the buffer is in state no. 4, and this is the state shown in Figure 6. If we read the remaining 2 elements (E and F), the buffer becomes empty and we get state no. 5. Note that *start* = 1 even though the buffer is empty, since the start of valid data can be anywhere in the underlying buffer.

Below we have declared the class `RingBuf` which is a ring buffer whose elements are of type `char`. In this assignment, you shall implement parts of this class.

```
class RingBuf {
private:
    char *buf;    // The underlying buffer
    int capacity; // Capacity of underlying buffer (max size)
    int start;    // Start of valid data
    int size;     // Size of valid data (0 if empty)
public:
    RingBuf(int capacity);
    RingBuf(const RingBuf &other); // copy constructor
    RingBuf(RingBuf &&other); // move constructor
    ~RingBuf();
    RingBuf& operator=(RingBuf rhs); // assignment operator, copy assignment
    void write(char c); // write a character to the buffer
    void write(string s); // write a string of characters to the buffer
    char read(); // read a char from the buffer
    string read(int count); // read a number of chars from the buffer
    string peek();
    friend void testRingBuf();
};
```

4a) Implement the constructor for `RingBuf`. The constructor should allocate the buffer `buf` with space for `capacity` number of elements. Remember to initialize `capacity`, `start` and `size` as described in the introduction.

4b) Implement the copy constructor.

4c) Implement the destructor.

4d) Implement the move constructor.

4e) Implement the assignment operator (copy assignment).

4f) Implement the function `void RingBuf::write(char c)` which writes an element `c` to the buffer. Remember to update both `start` and `size` as described in the beginning of the assignment.

4g) Implement the function `char RingBuf::read()` which reads an element from the buffer. If the buffer is empty, the function should throw an exception (a string). Remember to update `start` and `size` as described in the beginning of the assignment. You do not need to zero the element which was read.

4h) Implement the function `void RingBuf::write(string s)` which writes a string to the buffer.

4i) Implement the function `string RingBuf::read(int count)` which reads up to `count` number of elements from the buffer. If `count` is larger than `capacity` or `count` is `-1`, the function should read all the valid elements stored in the buffer.

4j) Implement the function `string RingBuf::peek()` which returns the contents of the buffer without removing elements. It is especially useful to be able to test that `RingBuf` behaves as expected. `peek()` returns a `string` with all the valid elements in the buffer, i.e. from `start` to (but not including) `start + size`.

4k) Testing is an important part of programming. Write a function `testRingBuf()` which checks that `RingBuf` behaves as expected. The test should create a ring buffer and perform the operations which were described in the table from the introduction. After each operation, you should check that the values of the member variables `start` and `size` are correct and that `peek()` returns the correct values. Check also the return value of `read()` when testing the read functionality. Use `assert()` as described in the appendix.

...---oooOooo---...

H IT2810, exam 2018

IT2810 Webutvikling H2018

i **Forside**

Institutt for Datateknologi og Informatikk

Eksamensoppgave i IT2810 Webutvikling

Faglig kontakt under eksamen: Trond Aalberg

Tlf.: 97631088

Eksamensdato: 18/12 2018

Eksamenstid (fra-til): 09-11

Hjelpemiddelkode/Tillatte hjelpemidler: E: Ingen hjelpemidler tillatt.













I vurderingen honoreres korte og presise svar.

Merk! Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

1 **Variabler deklarerert med var og let**

Forklar kort hvilket scope som gjelder for variabler som er deklarerert med nøkkelordet **var** og hvilket scope gjelder for variabler deklarerert med nøkkelordet **let**?

Skriv ditt svar her...













Format - | **B** | *I* | U | x_2 | x^2 | I_x |  |  |  |  |  |  |  |  |  |  |  | 

Words: 0

Maks poeng: 1

2 Arrow-funksjoner

Forklar kort hvordan arrow-funksjoner skiller seg fra vanlige funksjoner i Javascript, med tanke på **this**
Skriv ditt svar her...

Format • **B** *I* U x_2 x^2 I_x |            













Words: 0

Maks poeng: 1

3 CSS-grid og Flexbox

Forklar kort hva CSS-grid er og CSS-flexbox er. Beskriv hvilket problem/behov de løser og hva som skiller disse to løsningene.

Skriv ditt svar her...

Format • **B** *I* U x_2 x^2 I_x |            












Words: 0

Maks poeng: 1

4 SVG og HTML5 Canvas

SVG og HTML5 Canvas kan begge brukes til å lage interaktiv grafikk på websider. Forklar kort hva begge er og gi et eksempel på (og argumen) for en anvendelse hvor SVG er godt egnet og en anvendelse hvor HTML5 Canvas er godt egnet.

Skriv ditt svar her...

Format • **B** *I* U x_2 x^2 I_x |          Σ  












Words: 0

Maks poeng: 1

5 jQuery

Hva er selector-mekanismen i jQuery. Gi et eksempel og en kort forklaring.

Skriv ditt svar her...

Format • **B** *I* U x_2 x^2 I_x |          Σ  














Words: 0

Maks poeng: 1

6 Single Page Application

Hva kjennetegner en SPA (Single Page Application) ?

Skriv ditt svar her...

Format - | **B** *I* U x₂ x² | *I*_x |   |    |   |   |  |  |  | 














Words: 0

Maks poeng: 1

7 Responsiv webdesign

Hva er responsiv web design? Nevn forskjellige teknikker som brukes for å implementere responsiv webdesign.

Skriv ditt svar her...

Format - | **B** *I* U x₂ x² | *I*_x |   |    |   |   |  |  |  | 

Words: 0

Maks poeng: 2

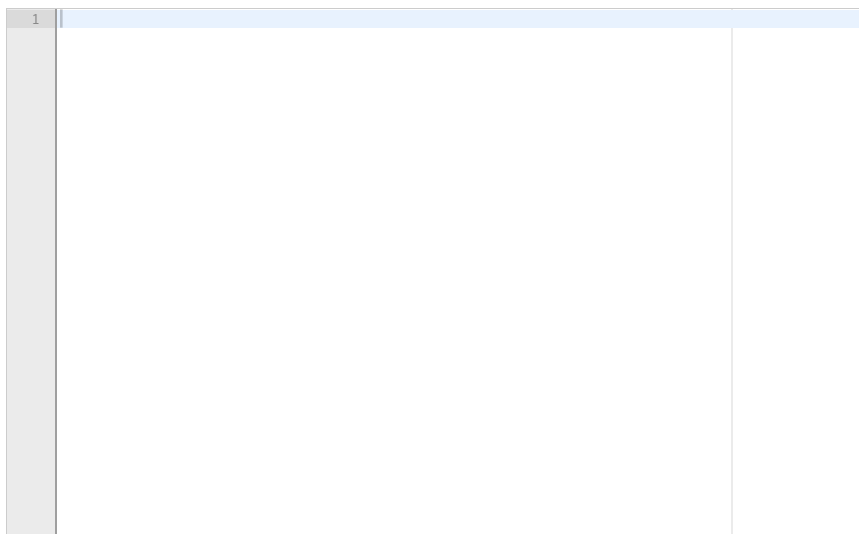
8 React

Lag en React-komponent kalt HelloWorld for et H1 element med teksten "Hello World!".

Komponenten skal ha følgende import-statement
`import React, { Component } from 'react';`

og skal kunne importeres av andre javascript-filer.

Skriv ditt svar her...














A large empty rectangular box for writing the answer. The box has a light blue header bar at the top. In the top-left corner of the main area, there is a small grey square containing the number '1'. The rest of the box is white and empty.

Maks poeng: 1

9 React props og state

Forklar kort hva props og state er i React

Skriv ditt svar her...

Format • **B** *I* U x_2 x^2 I_x             














Words: 0

Maks poeng: 2

10 React dataflyt

Forklar hvordan du må implementere dataflyt oppover i et React komponenthierarki.

Skriv ditt svar her...

Format • **B** *I* U x_2 x^2 I_x             












Words: 0

Maks poeng: 1

11 Web storage

Hvilken funksjonalitet tilbys gjennom HTML5 Web storage api'et (og det tilsvarende AsyncStorage api'et i React native)?

Skriv ditt svar her...

Format | **B** | *I* | U | x_2 | x^2 | I_x |  |  |  |  |  |  |  |  |  | Σ |  | 












Words: 0

Maks poeng: 1

12 React vs. React native

Beskriv hva som typisk kan gjenbrukes og hva som typisk ikke kan gjenbrukes hvis du skal gjøre om en React for web applikasjon til React native.

Skriv ditt svar her...

Format | **B** | *I* | U | x_2 | x^2 | I_x |  |  |  |  |  |  |  |  |  | Σ |  | 

Words: 0

Maks poeng: 1

13 State management

Hva er og hvorfor bruker vi state management som Redux og Mobx?
Gi eksempel på hvordan disse brukes i implementasjonen (dvs. vis litt kode).

Skriv ditt svar her...

Format - | **B** *I* U x₂ x² | *I*_x | | | | | | | |

Words: 0

Maks poeng: 2

14 Snapshot-testing

Forklar hva snapshot-testing er?

Skriv ditt svar her...

Format - | **B** *I* U x₂ x² | *I*_x | | | | | | | |

Words: 0

Maks poeng: 1

15 **REST/GraphQL**

Forklar kort hva REST er eller hva GraphQL er (velg en av disse). Vis eksempel.

Skriv ditt svar her...

Format - | **B** *I* U x₂ x² | *I*_x | | | | | |

Words: 0

Maks poeng: 2

16 **End to end testing**

Forklar hva end to end testing er?

Skriv ditt svar her...

Format - | **B** *I* U x₂ x² | *I*_x | | | | | |

Words: 0

Maks poeng: 1

