



Norwegian University of
Science and Technology

3D Representation of Complex Data Structures

Author(s)

Butt, Zohaib
Holt, Kent Wincent
Hauge Torkelsen, Eldar

Bachelor in Programming [Games | Applications]
20 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

20.05.2019

Supervisor

Kolloen, Øivind

Sammendrag av Bacheloroppgaven

Tittel:	3D Visualisering av Komplekse Datastrukturer
Dato:	20.05.2019
Deltakere:	Butt, Zohaib Holt, Kent Wincent Hauge Torkelsen, Eldar
Veiledere:	Kolloen, Øivind
Oppdragsgiver:	Norwegian University of Science and Technology
Kontaktperson:	Frantz, Christopher
Nøkkelord:	WebGL, Antlr, visualisering, programmeringspråk analyse, 3D, Git, MIT
Antall sider:	70
Antall vedlegg:	8
Tilgjengelighet:	Åpen

Sammendrag:	Kodebaser kan fort bli store og komplekse. Dette gjør at det kan ta veldig lang tid å få et overblikk over koden. Vi fikk i oppgave av Frantz, Christopher å implementere et system som kunne ta imot en Git URL og visualisere kodebasen som finnes i Git repositoryet, ved bruk av ulike farger og modeller. Dermed har vi laget en løsning som visualiserer kode i et 3D miljø på en oversiktlig og klar måte. Løsningen er en Web-App med en front-end i HTML/SASS/JS og en back-end API i Go med en Antlr-basert språk leser i Java. Vi har holdt en profesjonell utviklings metode, ved bruk av Scrum, Jira og Confluence, samt høy kode kvalitet og dokumentasjon av kildekoden og prosessen.
-------------	--

Summary of Graduate Project

Title:	3D Representation of Complex Data Structures
Date:	20.05.2019
Authors:	Butt, Zohaib Holt, Kent Wincent Hauge Torkelsen, Eldar
Supervisor:	Kolloen, Øivind
Employer:	Norwegian University of Science and Technology
Contact Person:	Frantz, Christopher
Keywords:	WebGL, Antlr, visualization, programming language, parsing, 3D, Git, MIT
Pages:	70
Attachments:	8
Availability:	Open

Abstract:	Codebases can quickly become large and complex. This makes it difficult to get an overview of the code. We received a task from Frantz, Christopher to consume a Git URL and visualize the codebase that resides within Git repository, by using different colors and models. Therefore we've produced a solution that visualizes the code in a 3D environment in a clear and concise way. The solution is a Web-App containing a front-end written in HTML/SASS/JS and back-end API written in Go with an Antlr-based language parser in Java. We followed a professional development method by the use of Scrum, Jira and Confluence as well as high code quality and documentation of both the source code and the development process.
-----------	--

Preface

We would like to thank our product owner Frantz, Christopher for making this project and choosing us as the development team for the project, but also help us improve the product throughout the bachelor. We would also like to thank our supervisor Kolloen, Øivind for great help in academic writing. A big thanks to the participants for user studies and other bachelor students that have helped us correct or shared information related to the thesis. We would also like to thank different lecturers and professors that have helped us to gain knowledge which has been used to develop the project, and finally thanks to McCallum, Simon and Farup, Ivar for making the L^AT_EX template which was used to write this thesis.

Contents

Preface	iii
Contents	iv
List of Figures	viii
List of Tables	ix
Listings	x
Glossary	xi
Acronyms	xvi
1 Introduction	1
1.1 Background	1
1.2 Subject area	1
1.3 Boundaries	2
1.4 Target group	2
1.4.1 Application users	2
1.4.2 Thesis readers	3
1.5 Motivation	3
1.5.1 Background competence	3
1.5.2 Knowledge to acquire	4
1.6 Constraints	4
1.6.1 Time Constraints	4
1.7 Roles	5
1.7.1 Group roles	5
1.7.2 Other roles	5
1.8 About Report	5
2 Requirements	6
2.1 Result Goal	6
2.2 Operational requirements	6
2.3 Learning outcome	7
2.4 Use Case	7
2.4.1 High level use case descriptions	8
2.5 Product backlog	10
2.6 Domain model	12
2.7 Risk assessment	13
2.7.1 Identification and project risk analysis	13
2.7.2 Risk mitigation strategy	13
2.8 Security requirements	14

3	Technical Design	15
3.1	Architecture	15
3.1.1	Front-end	16
3.1.2	Back-end	16
4	User Interface	21
4.1	Design	21
5	Development process	22
5.1	Development methodology	22
5.2	Choice of methodology	22
5.3	Execution	22
5.4	Sprint Overview	27
5.4.1	Sprint: hello world (02. feb - 13. feb)	27
5.4.2	Sprint: representation of complex data (14. feb - 18. feb)	28
5.4.3	Sprint: interactable (20. feb - 25. feb)	29
5.4.4	Sprint: metadata and coupling (26. feb - 04. mar)	29
5.4.5	Sprint: complex parsing and relation (05. mar - 11. mar)	30
5.4.6	Sprint: complex parsing and scope visual (12. mar - 18. mar)	31
5.4.7	Sprint: testing (19. mar - 02. apr)	32
5.4.8	Sprint: complex task and testing (02. apr - 08. apr)	33
6	Implementation	34
6.1	Front-end web development	34
6.1.1	Overall design implementation	34
6.1.2	Force-directed graph	35
6.1.3	Localization	38
6.1.4	Graphics	39
6.1.5	Interaction	41
6.2	Back-end	42
6.2.1	Overall design implementation	42
6.2.2	Processing server in Go	43
6.2.3	Language parsing in JAVA	44
7	Deployment	47
7.1	Dockerization	47
7.2	Deployment on SkyHigh with HOT	47
8	Testing and User Feedback	48
8.1	Unit testing	48
8.2	Code Quality	49
8.3	User studies	50
8.3.1	User categorization	50
8.3.2	Testing methodology	50
8.4	Results	51

8.4.1	Navigation and UI	51
8.4.2	Visualization and interaction	51
8.4.3	Proposed features	51
9	Discussion	53
9.1	Overall reflection and experience	53
9.2	Use of Jira	54
9.3	Quality metrics and measures	54
9.4	IaC	55
9.5	Perspective on antlr	56
9.5.1	Overall reflection and experience	56
9.5.2	Decreasing cyclomatic complexity in ANTLR	58
9.5.3	Use of regex	59
9.6	Perspective on REST and WebSocket	60
9.7	Perspective on Graphics systems	60
9.7.1	Overall reflection and experience	60
9.7.2	Choice of graphics framework	61
9.7.3	Choice of UI framework	61
9.7.4	Choice of Application Lifecycle Framework	61
9.7.5	Use of Force Directed Graph library	61
9.7.6	Graphics system refactoring	61
9.8	Perspective on FDG	62
9.8.1	Overall reflection and experience	62
9.8.2	Detailed experience on refactoring of FDG	62
9.9	"sloccount" vs "Kloc" vs "wc -l"	63
9.10	Platform for the product	63
9.11	Post-phoning/removing userstory allowing user to submit lexer files	64
10	Conclusion	65
10.1	Result	65
10.2	Future Work	65
10.3	Evaluation of group work	65
10.4	Ending	65
	Bibliography	66
A	Initial Project Description	71
B	Project Plan	73
C	Project Agreement	87
D	User study	91
D.1	Participation notes 1	92
D.2	Participation notes 2	94
D.3	Participation notes 3	96
D.4	Participation notes 4	98

D.5	Participation notes 5	100
D.6	Participation notes 6	102
D.7	Participation notes 7	104
D.8	Participation notes 8	106
D.9	Participation notes 9	108
E	Design	110
E.1	Initial visualization concept	110
E.2	Logo concepts	110
F	Jira exported decision log	111
G	Meeting Logs	114
H	Time log	147

List of Figures

1	Use case diagram.	8
2	Domain model.	12
3	System architecture.	15
4	Sequence diagram of Initial request.	19
5	Final UI layout.	21
6	Pull request discussion.	24
7	Velocity chart.	25
8	Scrum board for test sprint.	26
9	Burndown chart for sprint hello world.	27
10	Burndown chart for sprint representation of complex data.	28
11	Burndown chart for sprint interactable.	29
12	Burndown chart for sprint metadata and coupling.	29
13	Burndown chart for sprint complex parsing and relation.	30
14	Burndown chart for sprint complex parsing and scope visual.	31
15	Burndown chart for sprint testing.	32
16	Burndown chart for sprint complex task and testing.	33
17	Gravity equation based on logit.	36
18	Local to global index example.	36
19	Test summary of different Java ANTLR parser files.	48
20	Test summary of different Go API files.	48
21	Server network architecture.	56
22	CPP functiondefinition graph to depth 3.	58
23	Initial UI concept.	110
24	Logo for large and small format.	110

List of Tables

1	Group roles	5
2	Final backlog	11
3	Overview of risks	13
4	Risk analysis form	13
5	Risk mitigation	14

Listings

3.1	JSON structure from API	17
6.1	Use of Module pattern in stripped down FDG	34
6.2	Force calculation FDG	35
6.3	Get node function	37
6.4	Language option functionality	38
6.5	Linking function for calls	39
6.6	A snippet of raycast onMouseClick	41
6.7	Api routes	42
6.8	Websocket success response	43
6.9	Rule for memberdeclaration	44
6.10	If block example	44
6.11	Antlr C++ listener example	45
6.12	Antlr Java listener example	45
8.1	JSON structure for a valid test case	49

Glossary

- agile** [Development methodology](#) focused on adaptability and requirements or solutions to evolve through the collaborative effort of self-organizing and cross-functional teams and their end user. [1]. [22](#), [53](#)
- apache2** A free and open-source cross-platform web server [2]. [15](#), [16](#), [20](#)
- APIDOC** Generates a RESTful web API Documentation [3]. [23](#), [47](#)
- Automation Architect** The person who's responsible for design, implementation and analyzation of strategies for continuous deployments whilst ensuring high availability of product and pre-production stages [4]. [5](#)
- AVA** Futuristic JavaScript test runner [5]. [32](#), [49](#)
- back-end** Back-end is the data access layer of a software [6]. [xii](#), [15](#), [16](#), [27–33](#), [42](#), [43](#), [55](#), [61](#), [63](#)
- burndown** A graphical representation of work left to do versus time [7]. [viii](#), [27–30](#)
- CHEF** A configuration management tool [8]. [56](#)
- client** Computer in a network that uses the services provided by a [server](#). [xii](#), [15](#), [16](#), [18](#)
- Confluence** A collaboration software program developed and published by Atlassian. [24](#), [54](#)
- connacense metrics** A complexity measure for describing [coupling](#) within a program. [31](#), [55](#), [59](#)
- control flow** The order of which a programs instructions or statements are executed [9]. [1](#)
- coupling** Is the degree of interdependence between software modules [10]. [xi](#)
- cyclomatic complexity** A complexity metric based on number of conditional or paths through a program. [31](#), [44](#), [54](#), [59](#)
- development methodology** A framework that is used to structure, plan, and control the process of developing an information system. [xi](#), [22](#)
- DevOps** A set of practices that aim to improve connections between development and operations and shorten the development liefecycle while ensuring high quality. [3](#), [7](#), [22](#), [55](#)
- DevOps evangelist** The person who researches and pushes for DevOps concepts and methodologies [4]. [5](#)

Docker A computer program that performs operating-system-level virtualization known as containerization. [xii](#), [47](#), [56](#)

Docker Compose A tool for defining and running multi-container [Docker](#) applications. [23](#), [47](#)

dockerfile File containing instructions for automating [Docker](#) container setup. [23](#), [47](#)

documentation generator A programming tool that generates software documentation from a set of source code files. [23](#)

epic Large bodies of work that can be broken down into a number of smaller tasks [[11](#)]. [30](#)

Experience Assurance Professional The person who's responsible for ensuring focus on user experience in any new and old functionality. [5](#)

Facilitator The person who readies materials for meetings. [5](#)

formatter A computer program used for formatting. [23](#)

front-end The presentation layer of a software. [7](#), [18](#), [27–31](#), [33](#), [34](#), [42](#), [49](#), [55](#), [61](#), [63–65](#)

full-stack Anything that touches all levels between [client](#) and [back-end](#). [30](#)

Git Version control system. [3](#), [16](#), [24](#), [27](#), [43](#), [44](#), [54](#), [56](#)

Go Go or Go is a statically typed compiled language in the tradition of C, with memory safety, garbage collection, structural typing, and CSP-style concurrent programming features. [16](#)

Godoc Extracts and generates documentation for Go programs. [23](#)

halsted complexity A complexity metric based on number of Operands, Operators and the total number of usages of each of these. [31](#)

Hooke's law Describes a relationship between spring compression and force applied [[12](#)]. [35](#)

interface The means by which interaction or communication is achieved between systems. [xvi](#), [xvii](#), [15](#), [16](#), [28](#)

JavaScript A "call by sharing" scripting language often used for [client](#)-side web-applications. [xiii](#), [xvi](#), [15](#), [47](#), [49](#), [54](#), [60–62](#), [64](#)

Jenkins A self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. [[13](#)]. [47](#), [56](#)

Jira A proprietary issue tracking product. [24](#), [54](#)

JSDoc Markup language used to annotate JavaScript source code files. [23](#)

- JUnit** A simple framework to write repeatable tests [14]. 32
- key-value map** A data structure that maps keys to values. 38, 42
- Lead Interaction Designer** The person who's responsible for ensuring that interactions are user-friendly. 5
- Lead programmer** The person who's responsible for ensuring good code efficiency and quality. 5
- Lead Security Engineer** The person who gives recommendations towards security and enforces them as much as possible. 5
- linter** A tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs. 23
- localization** Translating a product into different languages or adapting a product to a country or region. 38
- logit** Inverse of the Sigmoid function [15]. 35, 36
- manhattan distance** The distance between two points measured along axes at right angles [16]. 13
- MongoDB** A document database storing data in [JavaScript Object Notation \(JSON\)](#) like documents [17]. 47
- namespace** A named scope that prevents symbols from being mistaken for identically-named symbols in other scopes. 16, 28, 30
- object-oriented programming** Object-oriented programming is a programming paradigm based on the concept of "objects", which can contain data, and code, in the form of procedures. 2, 34
- open-source** A term denoting that a product includes permission to use its source code, design documents, or content [18]. 3, 47, 53, 54
- OpenStack** A cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter[19]. xiv, 47, 55
- planning meeting** A process where product owner describes highest priority feature to the team. 25
- planning poker** A consensus-based, gamified technique for estimating [20]. 31, 54
- product backlog** An ordered list of everything that is known to be needed in the product [21]. xiv
- Puppet** A server configuration tool [22]. 56

quality metric A rule for quantifying some characteristic or attribute of a computer software entity. One of a set of techniques whose aim is to measure the quality of a computer program. [23]. 2

raycast The use of ray-surface intersection tests to solve a variety of problems in computer graphics and computational geometry. [24]. 41

Recorder The person who produces reports from meetings. 5

recursion A computer programming technique involving the use of a procedure, subroutine, function, or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first. xiv, 59, 63

recursive Of, relating to, or involving [recursion](#). 12, 13, 16

regular expression A sequence of characters that define a search pattern [25]. xvi, 59

Release Manager The person who's responsible for releasing production ready software [4]. 5

review meeting A process where [Scrum](#) team shows what was accomplished during the [sprint](#) [21]. 24, 25

scene graph A collection of nodes in a graph or tree structure which arranges the logical and often spatial representation of a graphical scene [26]. 39, 63

Scrum A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. [21]. xiv, 22, 53

Scrum Master The person that manages sprints, backlog. 5

server Computer in a network that is used to provide services to other computers in the network. xi

sigmoid Used to transform values on $(-\infty, \infty)$ into numbers on $(0, 1)$ [15]. 35, 36

SkyHigh [OpenStack](#) instance hosted by NTNU. 6, 47

software design pattern A reusable solution to a commonly occurring problem within a given context in software design. 25

SonarQube An open source platform to perform automatic reviews with static analysis [27]. 32

source code Any collection of code, possibly with comments, written using a human-readable programming language, usually as plain text [28]. 28

sprint A time constrained unit of work in [Scrum](#). viii, xiv, xv, 5, 22, 24, 25, 27–33

sprint backlog Set of [product backlog](#) items selected for the [sprint](#) [21]. 25, 26

- sprint retrospective** Meeting for planning improvements to be enacted during the next [sprint](#) [21]. [25](#)
- SQL** Structured Query Language used for Querying from certain database systems. [xv](#)
- SQLi-attack** [SQL](#) injection, input of executable [SQL](#) code that could modify a [SQL](#)-based database system. [14](#)
- story point** An abstract measure of effort required to implement a user story. [25](#)
- Sublime Text** A sophisticated text editor for code, markup and prose [29]. [49](#)
- syntax** The way in which linguistic elements (such as words) are put together to form constituents (such as phrases or clauses). [12](#), [13](#)
- taint propagation** Using data flow to determine what an attacker can control [30]. [55](#), [60](#)
- Tensorflow** An end-to-end open source platform for machine learning [31]. [21](#)
- Toggl** A time tracking app [32]. [54](#)
- use case** Usually a list of events, often used to describe the interaction between actors and systems. [7](#)
- Utility Technology Player** The person who's responsible of maintaining uptime of service and give inputs on security, quality and resource management [4]. [5](#)
- vetter** A computer program used for formatting source code. [23](#)
- WebGL** A JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins [33]. [7](#), [27](#), [60](#)
- Websocket** A communication protocol, providing three communication channels; "Open", "Close", "Message" over a single transmission control protocol connection. [15](#), [16](#), [19](#), [30](#), [31](#), [42](#), [43](#), [49](#), [60](#)
- white-box** Software testing approaches that examine the program structure and derive test data from the program logic [34]. [1](#)
- XSS-attack** Cross site scripting, input executable code that will run on other instances of the service. [14](#)

Acronyms

ALF	Application Lifecycle Framework. 61
ANTLR	ANother Tool for Language Recognition. 18 , 29 , 32 , 42 , 44–46 , 54 , 56 , 57 , 59 , 64
API	Application Programming Interface . 15 , 16 , 18–20 , 27 , 29–32 , 42 , 47 , 49 , 60 , 61 , 65
CI/CD	Continuous Integration and Continuous Deployment [35]. 55
CSS	Cascading Style Sheets. 15 , 47
DOM	Document Object Model. 41
FDG	Force Directed Graph. 16 , 28 , 31 , 34 , 35 , 37 , 61–63
GB	GigaByte. 6
GDPR	General Data Protection Regulation. 51
GUI	Graphical User Interface . 28 , 29 , 34 , 65
HOT	Heat Orchestration Template [36]. 47 , 55 , 56
HTML	Hyper Text Markup Language. 15 , 27 , 47
HTTP	Hypertext Transfer Protocol. 15 , 16 , 19 , 43 , 60
IaC	Infrastructure as Code. 7 , 23
IMRaD	Introduction, Material and method, Results and Discussion. 5
IT	Information Technology. 2 , 50
JAP	Java ANTLR Parser. 15 , 18 , 28–30 , 32 , 33 , 42 , 43 , 48
JSON	JavaScript Object Notation. xiii , 16 , 18 , 27 , 28 , 40 , 42 , 43 , 48 , 49
KLOC	Thousands of Lines Of Code. 6 , 63
LOC	Lines Of Code. 16 , 30 , 63
MVC	Model View Controller. 18
regex	Regular expression . 59
REST	REpresentational State Transfer. 15 , 19 , 27 , 30 , 43 , 60 , 65

UI User [Interface](#). [16](#), [18](#), [21](#), [60](#), [61](#)

UID Unique IDentifier. [19](#)

URI Uniform Resource Identifier. [27](#)

URL Uniform Resource Locator. [42](#), [43](#), [55](#)

VCPU Virtual Central Processing Unit. [6](#)

1 Introduction

1.1 Background

This project was requested by Frantz, Christopher from the Department of Computer Science at NTNU i Gjøvik. The project was requested because finding libraries or frameworks that would fit well for new software projects can be a time consuming and difficult process that might require [white-box](#) analysis. This project should make the [white-box](#) analysis easier by abstracting the code, representing it as a 3D graphical structure, whilst also giving information on complexity and quality of the project.

CodePark [37] and CodeCity [38] are two applications trying to solve the same type of problem with similar 3D solution, representing the code as houses or city blocks. CodePark visualize the codebase in park-like environment. Each class in the codebase is represented as an interactable cube. In the cube, all functions in each class are visible on the wall and upon a click, it highlights the function in the code, which is shown on a separate wall. CodeCity represents the code as navigable cities, where each building represents a class and each district represents a package.

The two examples differ from this project in how this project will represent connections related to [control flow](#) and the code not being represented as buildings or cities. This project will also represent different data structures such as vectors and maps.

1.2 Subject area

The software is meant to be used in a software development setting. Software development [39] includes many different processes such as system specification, software reuse, integration, development, testing and maintenance. The system will mostly help with the software reuse and integration part, but will also indirectly affect the other processes.

System specification deals with identifying what the system should do and the environment it acts in.

Software reuse takes those specifications and identifies preexisting systems that could be used instead of developing new software. This process often identifies libraries and frameworks that can be used, which is often recommended [40] in both large and small systems. Developers spend a lot of time debugging, securing, documenting, structuring or finding suitable libraries and frameworks. This can be frustrating and time consuming if there is no good documentation, source code or life cycle provided [41].

Software integration takes the preexisting systems or the software it integrates with and molds them to fit smoothly. The integration can vary in difficulty depending on the implementation of each software solution.

Software development creates the functionality that is not provided by preexisting solutions. It is recommended to have as little code complexity [42] as possible for readability [43] which can be hard for developers to maintain while programming and is partially reliant on how well the software integration went. Potential problems with

the preexisting systems are likely to emerge at this state as it is difficult to see limitation and bugs during the software integration phase. Finding bugs in this scenario is time-consuming [44] and might require the developers to do white-box testing on an unfamiliar system. Bugs that are not found during development will hopefully be found during software testing.

Software testing validates that the functionality works as expected with no side-effects. This can help spot bugs or shortcomings in the preexisting systems. If bugs are not found during software testing, then they are likely to persist through the life cycle of the software or until identified by the user.

Software maintenance deals with updating the software to remove issues found by users or through normal use. It also assures that dependencies stay up to date.

The problem this system will help with is giving an impression of the build and quality of libraries and frameworks that are considered by the software reuse process. The system will act as a lesser alternative to documentation or as a quick overview to the libraries or framework.

1.3 Boundaries

The system is not meant to help with software specification or the later stages of development after software integration. It is not meant to supersede or be an exhaustive substitution of documentation, but rather give an abstraction and a quick overview of the codebase. The abstraction will be limited to a 3D visualization and textual representation of metric and meta data. It will not change the software in question or improve the quality, but give the end user an insight into the software model and code quality. The project will not involve developing new [quality metrics](#), but use preexisting metrics that have been used by other developers and been proven to be help-full.

The system will not build, nor execute the codebase in question but rather do static analysis. There will neither be provided any test coverage nor execution of any shipped testing functionality.

1.4 Target group

Target groups for the project can be divided into two parts; Application users and thesis readers.

1.4.1 Application users

The application mainly targets three different settings or contexts;

- Academic - Students, lecturer or researchers studying [Information Technology \(IT\)](#).
- Professional - [IT](#) developers.
- Amateur - hobbyists looking into [IT](#).

Academic

In an academic setting, the application can be used by students to check for code quality and complexity of their hand-in or their work. However the students must be familiar to concept like [object-oriented programming](#) to understand the visual output from the system. These students are expected to have at least one programming related semester

for the system to be useful.

Lecturers can use the system in the lectures to explain the differences between loosely and highly coupled architectures or discuss code quality.

Professional and amateur

The main target groups of this web-application are programmers in both professional and amateur settings. Here the application will be used mainly by the lead programmer as well as developers and architects to decide whether a library or framework should be used in production by assessing its complexity.

Programmers that are in the learning phase might find this application useful because of its 3D visualization of the codebase which will help them to see the whole codebase as one structured and clear representation.

1.4.2 Thesis readers

Target group for the thesis is anyone interested in knowing about the development process, motivation or implementation of the project. First and foremost anyone involved in the grading process, but also any developer who would take the project further as an [open-source](#) project or other derived work. This thesis is a natural follow-up after reading the documentation provided in the [Git](#) repository.

1.5 Motivation

The project description, as shown in appendix [A](#), indicated great potential and was interesting. It allowed for flexibility and depth, by spanning a variety of technologies and subject areas. It mentioned web-technologies, language parsing, complex data structures, visualization with interaction and container technologies. For the group it was a good mix of familiar technologies for further exploration and unfamiliar concepts to study. The project would allow for a practical application of [DevOps](#) and system architecture which the group had a more theoretical understanding of. It would also explore language parsing that the group has a very limited knowledge of and associate with low-level, complicated or compiler related subject areas. Parsing would allow for a deep dive into niche aspects of C++, Java or other programming languages. In addition the project was about static analysis that could help to improve understanding of code quality. The main motivation was the high learning outcome.

1.5.1 Background competence

The groups knowledge that was considered relevant for the project included:

- Operating systems - All members had theoretical knowledge through a course [\[45\]](#) and practical experience with Linux as main driver for several years.
- Architecture and static analysis - All members had a theoretical understanding through the Software engineering course [\[46\]](#) and Software security course but lacked more in-depth practical experience.
- Visualization and interaction - All members had experience through the Graphics programming course [\[47\]](#) and the Game programming course [\[48\]](#).
- Complex data structures - All members had theoretical experience through math for programming [\[49\]](#), Math for computer science course [\[50\]](#) and Algorithmic

methods course [51].

- Web-development - All members had experience through Cloud technologies course [52]. Besides, individual members had experience through Web technology course [53], Software security course [54] project or learning assistant task.

1.5.2 Knowledge to acquire

- Static analysis
- Language parsing
- Quality metrics and measures
- Complex data representation
- Client-side technologies

1.6 Constraints

Legal Constraints

- The resulting system will not be in violation of general data protection regulation. [55]
- The system will not be in violation of Norwegian copyright act.

Technological Constraints

- The system should be cross platform.
- The system should be dockerized.
- The system should have a REST API.

1.6.1 Time Constraints

Project planning started 5th Jan 2019.

- The system should be finalized by 16th May.

1.7 Roles

1.7.1 Group roles

Group roles	
Member	Role
Kent Wincent Holt - 473209	The Group leader Lead programmer Software developer/tester Release Manager
Eldar Hauge Torkelsen - 473180	DevOps evangelist Scrum Master Automation Architect Software developer/tester Lead Security Engineer Recorder Lead Interaction Designer
Zohaib Butt - 473219	Software developer/tester Experience Assurance Professional Facilitator Utility Technology Player

Table 1: Group roles

1.7.2 Other roles

The other roles involved in the thesis are supervisor and product owner.

- The supervisor role is filled by Kolloen, Øivind and will be responsible for providing the feedback, insights and corrections on both report structure and content.
- The product owner role is filled by Frantz, Christopher and will be responsible for providing wanted criteria. Discussing these criteria regarding how time consuming, how demanding they are and planning upcoming [sprints](#).

1.8 About Report

The report contains a lot of acronyms and expressions as these are commonly used in a development setting, but they might not be intuitive and therefore a glossary has been included to solve this problem. The structure of the report is based on the [Introduction, Material and method, Results and Discussion \(IMRaD\)](#)-model and provided template for bachelor thesis by NTNU i Gjøvik.

2 Requirements

The group have divided the requirements in three sections: result goals, operational requirements and learning outcomes. Result goals defines what the group expect to deliver as the end product. Operational requirements describe what the team expect from the system after a set amount of time and are listed to give an indication of the intended quality of the final system. Learning outcomes describe what the team wants to learn during development and thesis writing.

2.1 Result Goal

- Program must, at a minimum, be able to abstract and visualize language version up to Java 11.0.2 and C++17.
- Code visualization must be in 3D.
- Program must give an indication of the quality of the software code.
- Program must be able to take a link to a git repository to use as basis for visualization.
- Program must be able to show potential execution paths or program flow through the program.
- Program must have a public facing API for developers.
- Program must be dockerized.
- The user must be able to interact with the 3D visualization to show implementation of data structures.
- The system must present complexity metrics to the user.
- The system must give an indication on where complexity arises.

2.2 Operational requirements

The following operational requirements assumes a server equivalent of a [SkyHigh](#) instance of flavor c1.tiny with 16 [GigaByte \(GB\)](#) of ram, 40 [GB](#) of disk space and 8 [Virtual Central Processing Units \(VCPUs\)](#).

1. Quantitative goals for the system:
 - The system must handle 30 concurrent users with repository of less than 10 [Thousands of Lines Of Code \(KLOC\)](#) while using less than 10 minutes of processing time.
 - The system must be able to handle a peak of at least 60 users.
 - The system must be able to handle repositories of less than 100 [KLOC](#).
2. Qualitative goals for the system:
 - The system must help developer(s) gain overview and an understanding of the code base.
 - The system must let the user easily navigate through the visualization.
 - The system must be naturally intuitive and easy-to-use for new users.

- The system must be accessible to people with minor disabilities, like color-blindness.

2.3 Learning outcome

- Learn in depths [WebGL](#) technology.
- Learn professionalism in web development.
- Understand and follow [DevOps](#) methodology throughout the project.
- Learn in depths dockerizing containers.
- Learn [front-end](#) web frameworks.
- Learn about different code quality metrics for static analysis.
- Learn about professional web architectures.
- Learn about [Infrastructure as Code \(IaC\)](#).

2.4 Use Case

To achieve more detailed view of the actual systems functionality that need to be implemented, it is important to visualize the system from users perspective without focusing on implementation details. To do this, [use cases](#) and a use-case diagram are used as they show users needs.

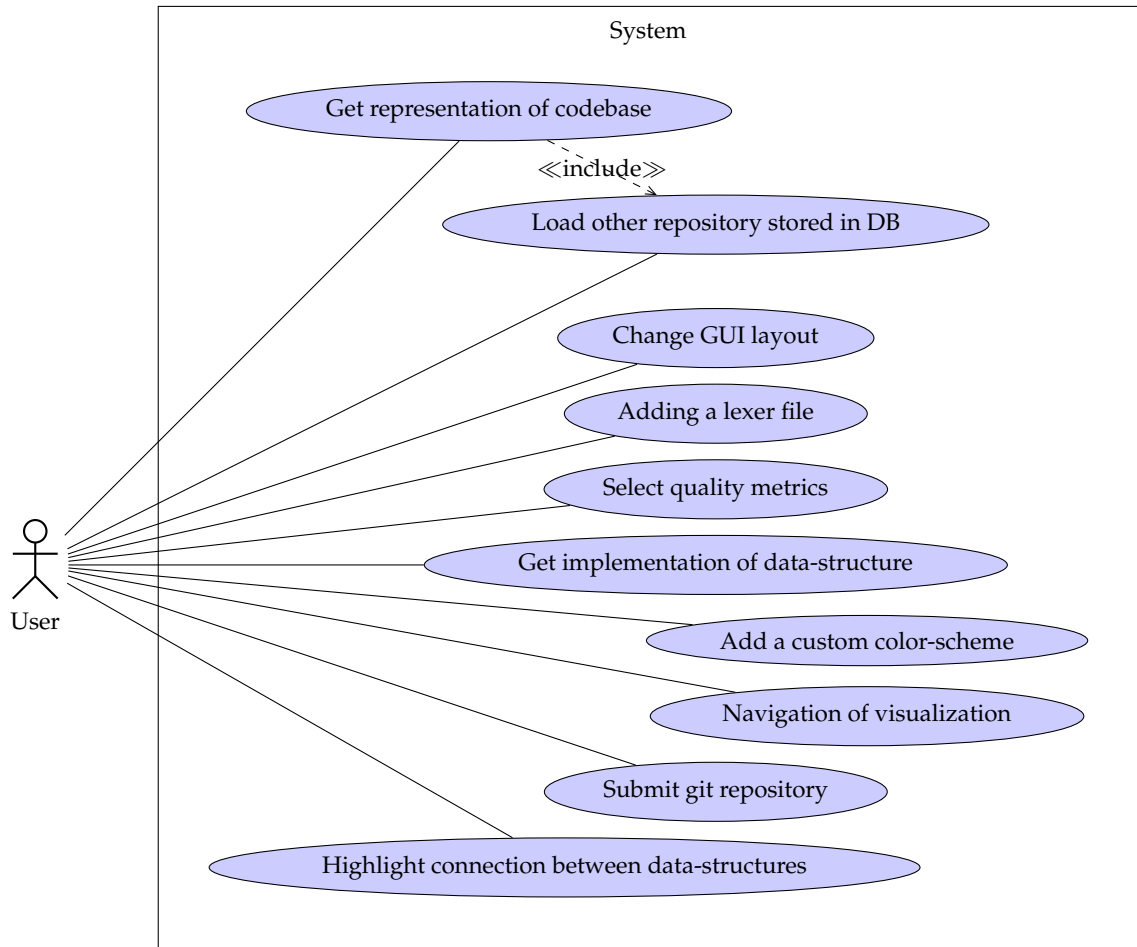


Figure 1: Use case diagram.

2.4.1 High level use case descriptions

Use case name: Change GUI layout

Actors: User

Target: Target is to change the GUI layout in the application.

Description: User can change the GUI layout by dragging and scaling windows. User can also add or remove a window of same type or any other type.

Use case name: Add a color-scheme

Actors: User

Target: Target is to change the color-scheme used in the application.

Description: User can change the color-scheme of application by specifying a set of colors from a palette on the home page.

Use case name: Add a lexer file

Actors: User

Target: User can add new lexer file to the system.

Description: User can submit a new lexer files on the home page by clicking on the "upload lexer file" and choosing the file from a remote machine.

Use case name: Select quality metrics

Actors: User

Target: User can select a specific quality metric.

Description: User can choose between different quality metrics such as "connascence measure", "cyclomatic complexity" and "Halstead complexity measures" on the quality metrics window. The chosen metrics will then be used for analysis of the submitted codebase.

Use case name: Submit git repository

Actors: User

Target: User can submit the git repository

Description: User can submit the git repository by adding repository link on the text field with name "Git Repository" and click submit to store repository in the database. The field can be found at home page.

Use case name: Load other repository stored in DB

Actors: User

Target: User can load the repositories submitted by other users.

Description: Database has multiple repositories stored by different users. These repositories will be shown in the window named "Repositories". A user can click on available repositories to visualize it.

Use case name: Get implementation of data structure

Actors: User

Target: User can see the implementation of chosen data structure.

Description: This functionality is required to have at least one entry in the database so that the system can get the representation of codebase. Once the visualization of codebase is shown, user can choose between visualized data structures to see the implementation on a window named "Implementation".

Use case name: Navigation of visualization

Actors: User

Target: User can navigate in the 3D environment to see visualized models.

Description: This functionality requires a submitted repository so that the representation of codebase can be shown. The navigation includes rotation, translation and zoom. The user can navigate in 3D environment using mouse or keyboard.

Use case name: Highlight connection between data structure

Actors: User

Target: Highlighting relations between data structures.

Description: This functionality requires a submitted repository so that the representation of codebase can be shown. Once the codebase is visualized, the user can choose between different data structures visualized to highlight it's links with other data structures. Highlighting connections will brighten the color of all the links and connected data structures, but also dull the color of all the unconnected data structures.

Use case name: Get representation of codebase

Actors: User

Target: Visualizing the codebase in 3D environment.

Description: The user starts at the homepage where the git repository to be visualize needs to be submitted. Once the requested git link is submitted and stored in database, user is taken to loading page where user can see the status of processing codebase file from back-end server. After the processing is done user is taken to 3D visualization page where the visualization will show the different data structures represented as a shape and a color.

2.5 Product backlog

The final backlog can be seen in table 2. It show a more detailed view than what is described in the use cases.

Summary	Issue key	Issue Type	Status
Make antlr scope sensitive	COD-42	Story	Done
As a developer i would like the duplicated content from the parsed output to be merged.	COD-76	Story	In Progress
As a user i would like to see the cyclomatic complexity of the codebase being visualized.	COD-74	Story	In Progress
As a user i would like to only see the names of the data structures within some radius of the mouse pointer.	COD-75	Story	To Do
Save output from java parsing in the database	COD-59	Story	To Do
As a user I would like to see what functions are tightly coupled in the 3D representation	COD-8	Story	To Do
Write tests for model/controller repo.go	COD-62	Story	Pending review
As a user i would like to navigate the 3d representation.	COD-10	Story	To Do
Write ava test for repo/list	COD-66	Story	To Do
Write ava test for repo/(repold)/file/read	COD-68	Story	To Do
Write ava test for repo/(repold)/initial	COD-67	Story	To Do
Show things belonging in a scope inside its scope in the visualization	COD-57	Story	Done
As a user i would like to see classes parsed for both java and cpp target	COD-56	Story	Done
Write tests for Cpp and Java ParserFacade.java	COD-69	Story	Done
Write test for MongoDB.go	COD-73	Story	Done
Write tests for java Models	COD-72	Story	Done
Write ava test for repo/add	COD-65	Story	Done
Write unit testing and apitests for java and golang	COD-61	Epic	To Do
Write tests for typeLogger.go	COD-64	Story	Done
Write tests for validation.go	COD-63	Story	Done
Write tests for CodeSnippetController	COD-60	Story	Done
Implement level of detail (LOD) on force directed graph to hide details when zoomed out	COD-58	Story	To Do
As a developer i would like the API endpoint extended to give information about what functions call what.	COD-9	Story	Done
Get what functions are called by other function	COD-47	Sub-task	Done
Research quality metrics and existing solutions.	COD-51	Story	Done
Add websocket to front-end	COD-37	Story	Done
Add websocket for initial request frontend	COD-50	Sub-task	Done
Setup frontpage with information about the project	COD-34	Story	Done
Add websocket to back-end	COD-36	Story	Done
Add websocket for initial request backend	COD-49	Sub-task	Done
As a user i would like to view or hide the implementation of a function or class by clicking it.	COD-11	Epic	In Progress
Parse lines of code in a file	COD-45	Sub-task	Done
As a user I would like to see updates on loading state	COD-35	Epic	In Progress
Add websocket for add request frontend	COD-39	Sub-task	Done
Add websocket for add request	COD-38	Sub-task	Done
Setup window for displaying code to show on response from backend	COD-30	Story	Done
Parse number of namespaces in a file	COD-46	Sub-task	Done
Parse number of functions in a file	COD-43	Sub-task	Done
Parse number of classes in a file	COD-44	Sub-task	Done
Send request to api when click on 3Dview function object.	COD-31	Story	Done
Extended API endpoint to give enough to represent simple and multidimensional variables.	COD-7	Story	To Do
Setup imgui	COD-28	Story	Done
Find a way for antlr to parse only scope.	COD-29	Story	Done
Replace Java Model based structure with tree to allow for better scope handling.	COD-33	Story	To Do
Make API send implementation in given start/end interval.	COD-25	Story	Done
Update function response	COD-32	Sub-task	Done
As a user i would like to submit a git repository link for the system.	COD-6	Story	Done
As a user i would like to see a representation of a simple function	COD-3	Story	Done
As a developer i want an API endpoint that gives me enough to represent a basic function.	COD-4	Story	Done
As a developer i would like an API endpoint giving metadata about the project.	COD-13	Epic	To Do
As a developer i would like the API endpoint to include comment coverage and test coverage	COD-15	Epic	To Do
As a user I want to submit hello world repository and visualize my code in 3D	COD-2	Epic	To Do
Write tests for Cpp and Java Lstnr_initial.java	COD-71	Story	To Do
Write tests for Cpp and Java ExtendedListener.java	COD-70	Story	Pending review
As a user i would like the data structures in the 3D visualization to group together with related structures	COD-19	Epic	In Progress
As a user i would like to submit a lexer file for defining language syntax	COD-23	Story	To Do
As a developer i would like coding language for analysis to be based on provided lexer file.	COD-24	Story	To Do
Add logger for error loggin on backend	COD-48	Story	Done
Research connasence quality metrics and find existing solutions	COD-54	Sub-task	Done
As a user i would like to highlight connections from selected node.	COD-12	Story	To Do
As a user i would like to resize windows	COD-20	Story	To Do
Research cyclometric complexity quality metric and find existing solutions	COD-53	Sub-task	Done
As a user i would like to display code in a seperate window	COD-21	Story	Done
As a user i would like to see what namespace datastructures are in.	COD-17	Story	Done
Add IMGUI files for displaying available repositories	COD-40	Story	Done
Add api endpoint for listing saved repositories	COD-41	Story	Done
Frontend refactoring	COD-27	Story	Done
As a user i would like to be notified if my browser does not support WebGL or this application	COD-5	Story	Done
As a user i would like to change the colorscheme.	COD-18	Story	To Do
As a user i would like code to be displayed with syntax highlighting	COD-22	Story	To Do
As a user i would like to see test coverage and comment coverage for Golang	COD-26	Story	To Do

Table 2: Final backlog

2.6 Domain model

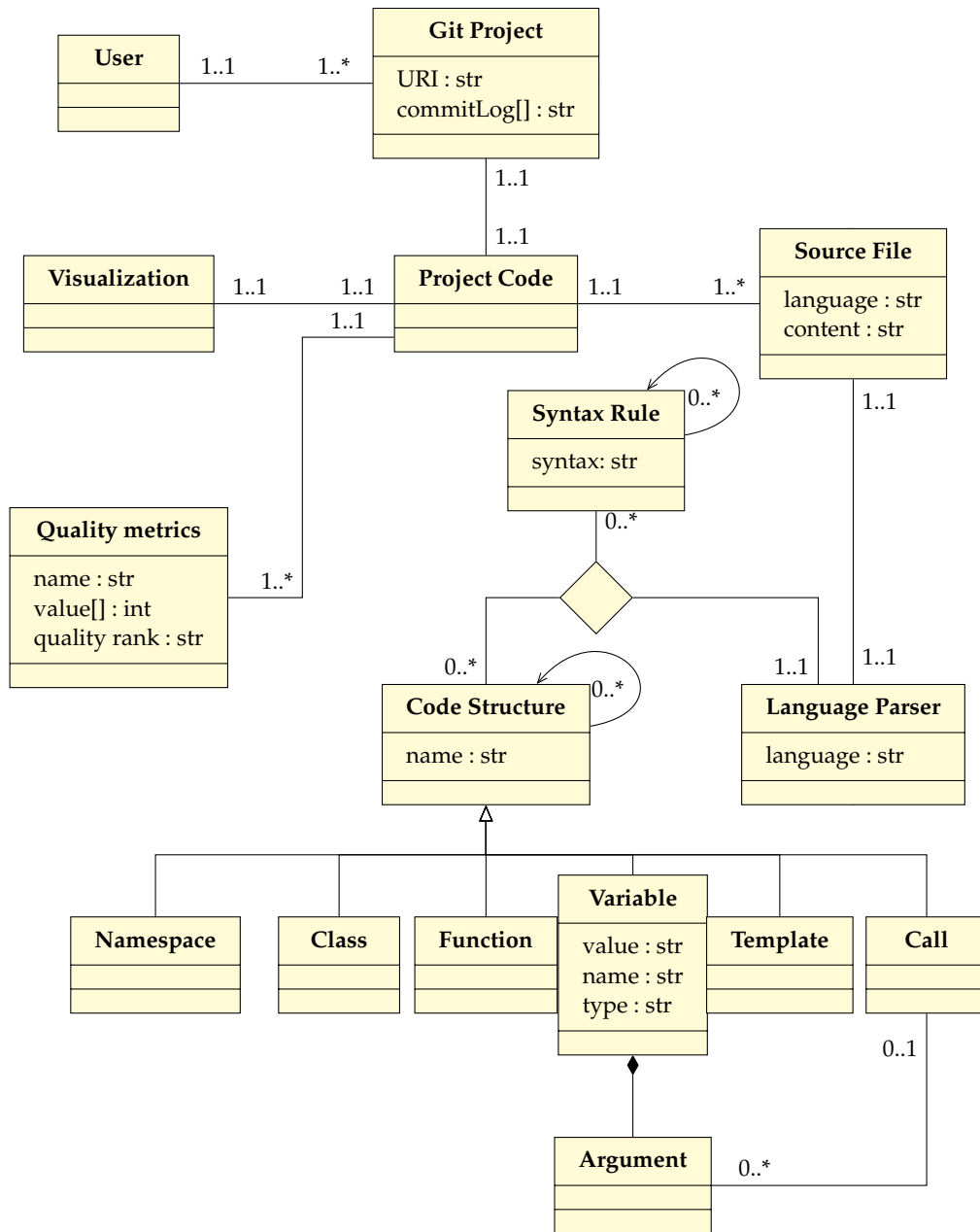


Figure 2: Domain model.

Figure 2 shows a crude overview of the subject area relating to the project. The code structure is language dependent and recursive based on the syntax rules relating to the structure. The relationship between a call, variable and parameter relates to how a function-call can have parameters that define which overloaded version of a function is being called. This relationship is given as an example but there are many other similar relationships that have not been added for clarity, like how variable types such as classes can define the scope a function is being called from. This is a relationship be-

tween functions being defined as part of a class, variables being instances of said class and calls being called on said variable.

Syntax rules are also **recursive** as they can be defined by a set of rules or a character sequence.

2.7 Risk assessment

The initial task description and product owner did not give any specific security requirements and has therefore done a risk assessment to identify any shortcomings that might happen and how to deal with them. This risk assessment will be a part of the base for the security requirements.

2.7.1 Identification and project risk analysis

The table 3 contains possible risks based on [37] and compared with the system requirements. The probabilities and effects are estimates done by the core team. The priority is based on the minimal **manhattan distance** from top right corner of table 4.

Id	Risk Type	Possible Risk	Priority	Probability	Effect
1	Organization	Losing important data	7	Low	Significant
2	Technology	Technology components aren't scalable	4	Medium	Severe
3	Technology	Insufficient security for the system	5	Medium	Significant
4	Technology	Application instability	5	Low	Severe
5	Organization	Team members are unavailable for longer period of time	6	Medium	Significant
6	Requirement	Discovering problems in requirements at delivery	1	High	Severe
7	Estimate	Project size is under estimated	3	High	Significant
8	Organization	Learning curves lead to delays	2	Very High	Moderate
9	Technology	Integration testing environments aren't available	9	Low	Minor
10	Organization	Fail to follow software methodology	8	Medium	Minimal

Table 3: Overview of risks

	Minimal	Minor	Moderate	Significant	Severe
Very High			8		
High				7	6
Medium	10			3, 5	2
Low		9		1	4
Very Low					

Table 4: Risk analysis form

2.7.2 Risk mitigation strategy

Table 5 shows the mitigation strategy for the project risks with highest priority and should give an overview of mitigations that should help with most of the identified risks, as several can be mitigated with similar strategies.

Priority	Risk	Mitigation
1	Discovering problems in requirements at delivery	<ul style="list-style-type: none"> • Use an agile development process • Perform regular user testing
2	Learning curve lead to delay	<ul style="list-style-type: none"> • When a new technology is to be used, a workshop for exploring it will be held
3	Project size is under estimate	<ul style="list-style-type: none"> • Use an agile development process • Have regular meetings with product owner
4	Technology components are not scalable	<ul style="list-style-type: none"> • Use IaC and code for modularity
5	Application instability	<ul style="list-style-type: none"> • Unit test coverage of all important functions • Use DevOps with live server throughout the development
6	Team members are unavailable for longer period of time	<ul style="list-style-type: none"> • Focus on reviewing git pull requests well • Good communication amongst developers
7	Losing important data	<ul style="list-style-type: none"> • Use cloud storage • Additional data storage security systems • Have and take regular backups

Table 5: Risk mitigation

2.8 Security requirements

- Sanitization of input from external sources within visualization and field for repository link submission to mitigate [SQLi-attack](#) and [XSS-attacks](#).
- The application will not in any way record or process any data that can be used to identify usage patterns for individual or corporation.
- Application will not store nor expose sensitive personal or corporate project information.
- The submitted repositories that are stored, are only stored for visualization of the codebase and will not at any point be built by the application.
- Any submitted repository will not be used for any other purpose than for visualization and quality assessment.
- Extensive testing to mitigate instabilities.

3 Technical Design

3.1 Architecture

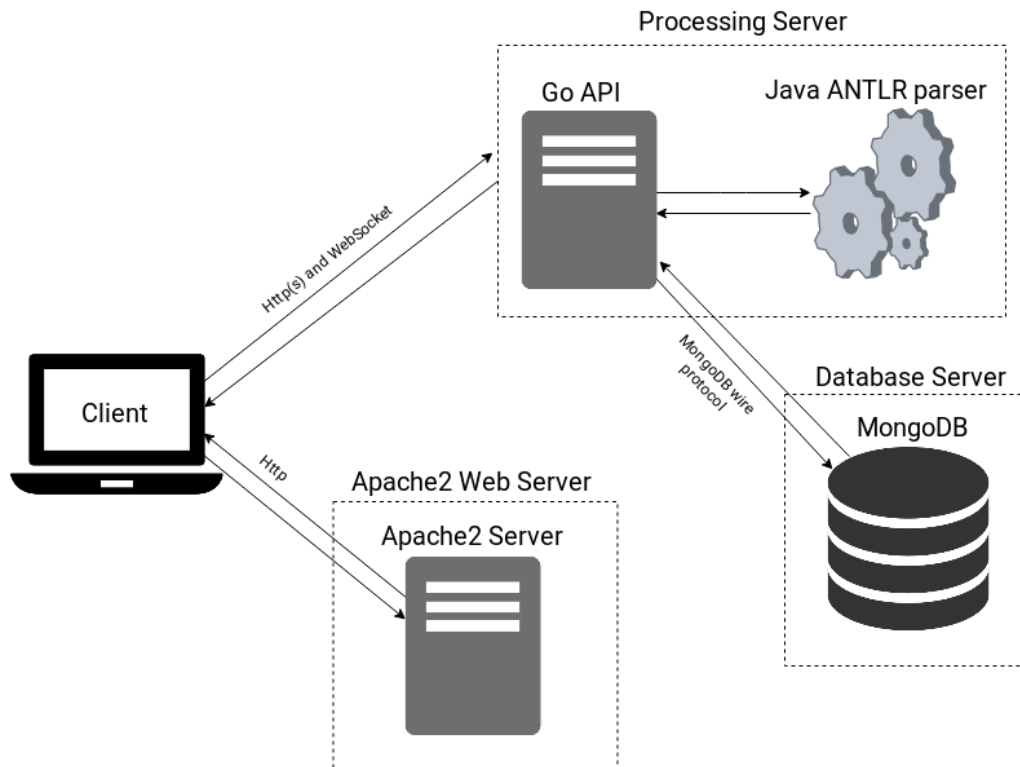


Figure 3: System architecture.

As the figure 3 shows, the team divided the different components of the system. Initially the team had planned to implement the solution as a modern **REpresentational State Transfer (REST)**-full **API**, where **client** sends request to processing server for data to be handled. **Client** uses **Hypertext Transfer Protocol (HTTP)** and **Websockets** to communicate with **Go API**, the reason **Websockets** are used to update **client** on lengthy **back-end** processes. **Websockets** can send multiple responses to the one sending request, this way we were able to show the status of processing to the **client**.

The processing server consists of two components **Go API** and **Java ANTLR Parser (JAP)**. **Go API** is mainly a **REST**-full **API** but with some functionality implemented with **Websockets**. **Go API** is the common interface that **clients** communicate through, to store or receive data from the database server. The **JAP** is a stand-alone helper application is used by the **Go API** to parse submitted project files. **Apache2** web server is used to send **Hyper Text Markup Language (HTML)**, **Cascading Style Sheets (CSS)** and **JavaScript** to

the [client](#) through [HTTP](#) protocol. Database server is used by Go [API](#) to update and fetch the data based upon [clients](#) request.

3.1.1 Front-end

The [User Interface \(UI\)](#) from is given from a static [Apache2](#) server and populated by a dynamic [Go](#) server. The front-end consists of two separate parts; project information and codebase visualizer.

The codebase visualizer is where the functionality lies. It communicates with the [API](#) to acquire a language independent representation of the codebase and getting parts of the codebase source. It expects the [API](#) to serve [JSON](#) in a similar format to listing [3.1](#). The listing shows the content of a [Websocket](#) response where different parts of the response can be explained as below:

- statuscode and statustext - Refers to a [HTTP](#) status to create a more uniform interface with [Websockets](#) and normal [HTTP](#) requests.
- body - The requested data with what is required to represent the codebase and information on the completeness of the parsing.
- files - Represents a list source files in the repository.
- functions, namespaces, classes, variables - Code blocks that the [client](#) expect to contain code blocks that can be [recursive](#) and used for visualization.
- call - A function call or assignment found in function code-block.
- scope - Shows if a call is executed on a variable.

Although what type of code-block can be within a another code-block might depend on the language. The [client](#) application allows for any code-block within any other and relies on the structure from the [API](#) to be correct. This is done to make the application as language independent as possible.

The [API](#) result is parsed to create the visualization with scopes, identifiers and relations. The representation is structured using a [Force Directed Graph \(FDG\)](#) to group together related structures and push unrelated structures further apart. Each scope goes through the [FDG](#) separately to ensure that the scope is maintained in the representation whilst the scopes internal relations are based on the calls. The calls are connected to the closest structure found to the function being called. The call might not always be connected to the actual callee function as the callee function might be outside the repository or can not be identified due to the parser limitations.

In addition the codebase visualization has a textual description of project metadata such as [Lines Of Code \(LOC\)](#) and number of structures like [namespaces](#), classes and functions, that were found.

The project information part describes the project, refers to the [Git](#) repository and describes the limitations of the program.

The limitations include unsupported browsers and limitations created by the incomplete parsing or unhandled code constructs.

3.1.2 Back-end

The processing server is the main server in the [back-end](#). It is responsible for handling the request from the [client](#) and send a response with a meaningful [HTTP](#) status code.

```
0 {
1   "statusCode": 200,
2   "statustext": "OK",
3   "body": {
4     "fileCount": 2,
5     "id": "5cc9c475b7fa706ccb4700f1",
6     "parsedFileCount": 1,
7     "result": {
8       "files": [
9         {
10          "parsed": false,
11          "file_name": "5cc9c475b7fa706ccb4700f1/README.md",
12          "linesInFile": 0
13        },
14        {
15          "parsed": true,
16          "file_name": "5cc9c475b7fa706ccb4700f1/main.cpp",
17          "functions": [
18            {
19              "name": "main()",
20              "declrator_id": "main",
21              "return_type": "int",
22              "function_body": {
23                "calls": [
24                  {
25                    "identifier": "printHelloWorld()",
26                    "scopes": [
27                      {
28                        "identifier": "World",
29                        "type": "namespace"
30                      }
31                    ]
32                  }
33                ]
34              },
35              "start_line": 15,
36              "end_line": 19
37            }
38          ],
39          "namespaces": [],
40          "linesInFile": 18
41        }
42      ]
43    },
44    "skippedFileCount": 0,
45    "status": "Done"
46  }
47 }
```

Listing 3.1: JSON structure from API

JAP uses ANOther Tool for Language Recognition (ANTLR) for language processing. ANTLR is a language recognition tool that uses a grammar file to assign a lexer token to each rule and then matches the token with a character sequence in the parsed file. ANTLR provides a listener for each rule in the grammar file. Once a rule matches a lexer token, it will callback to the respective listener. With this setup it was straight forward to use ANTLR, but it took a lot of time to understand the grammar files. Grammar files are created by different contributors which means names given for rules in the grammar are dependent on the contributor's choice and made it very hard for us to find rules for very specific language functionalities.

Go API uses parts of Model View Controller (MVC) pattern. MVC pattern is widely used within web development because it excludes the coupling between UI and logic. Go API uses the model and controller parts of MVC and view is managed by front-end. Controller in Go API is responsible of handling validation, error handling and business logic whilst models represents abstraction of data model and encapsulates database handling.

JAP is designed to take in two arguments; file path and target language used in the file. In return JAP will output the parsed code in JSON format to the standard output channel. The JSON format was necessary so that parsed code would be the same for both C++ and Java targets.

As figure 4 shows, Go API is responsible for handling requests from client, validation, execution of JAP and handling database. For the initial request from the user, the Go API will first check the database if project has been parsed previously and return the cached codebase. If no caching has been done, then the Go API executes the JAP and passes the file to be parsed along with a language identifier based on file extension.

If the project to be parsed has multiple files, then the Go API will check what files in the repository are supported by JAP and will only send those files as an argument, one at a time. This helped in limiting the ANTLR processing time. Once the parser output is received in Go API, it will construct and add the output into project JSON structure. Once all supported files are parsed, the project JSON structure will be sent as a response to the client.

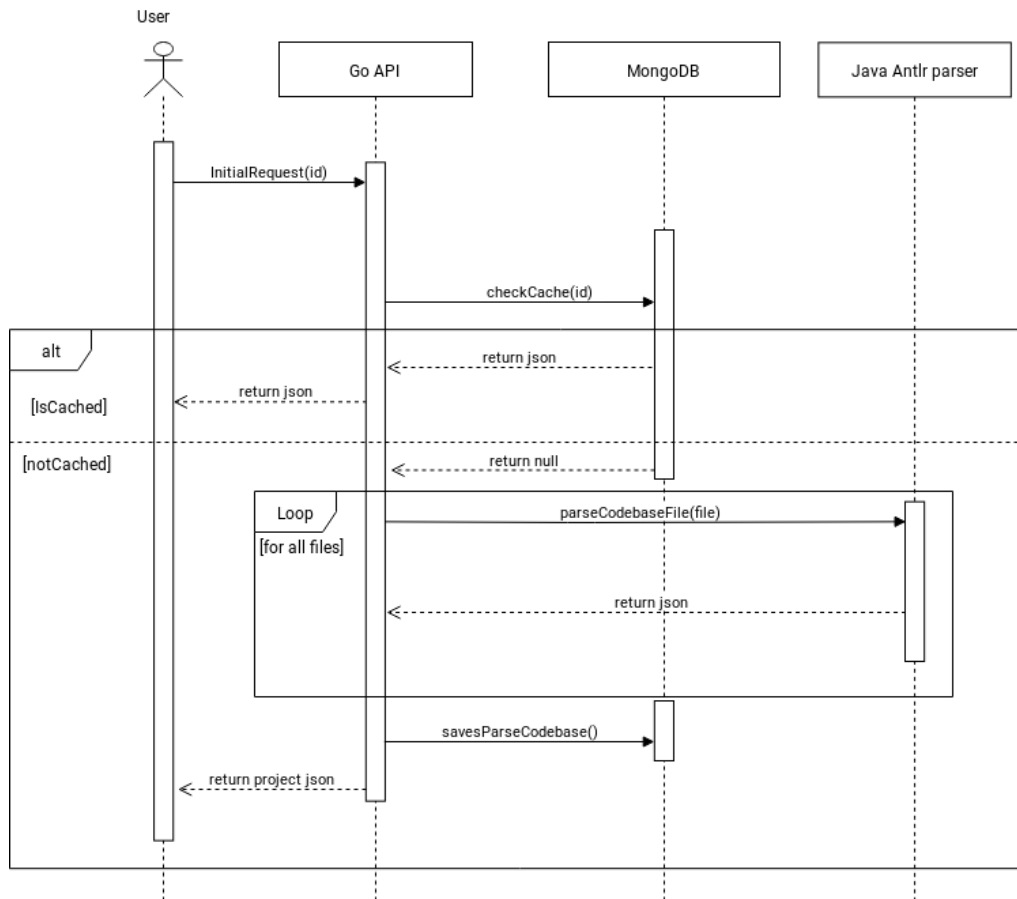


Figure 4: Sequence diagram of Initial request.

WEB API

At the start there was a requirement to have a [REST](#)-based [API](#) to receive information about a given project. We therefore came up with the following structure:

- `/repo/add` - Addition of a repository to the servers, this will, if not already cloned, clone it and return a new [Unique Identifier \(UID\)](#) else it returns the existing [UID](#).
- `/repo/list` - List all repositories that has been added to the server. So the user can see what's been parsed before.
- `/repo/{repoId}/initial/` - The initial request for the visualization. It is used to fetch the project structure and complexity measures. In the beginning this was a [HTTP](#)-method and since parsing a repository takes a long time, it could result in the user getting annoyed that no feedback on progress is returned. It was decided then that the initial [API](#) endpoint would be upgraded to a [Websocket](#) that can maintain a connection and send multiple packets more naturally than basic [HTTP](#)-methods. This made us drift from the requirement of a full [REST](#)-based [API](#). To re-fulfill this requirement we could re-implement the original endpoint and serve it together with the [Websocket](#).

The final result of the parsing is also cached for a quicker response, but the caching

mechanics is very crude in the sense that it doesn't update the cached data even if the submitted repository has changed.

- `/repo/{repoId}/file/read/` - Used to fetch any implementation at a specified location and size, in a given file.

A lot of focus was put on documentation, the group decided therefore to use [API Doc](#) for documenting the endpoints. The generated documentation is hosted using an [Apache2](#) web server.

4 User Interface

4.1 Design

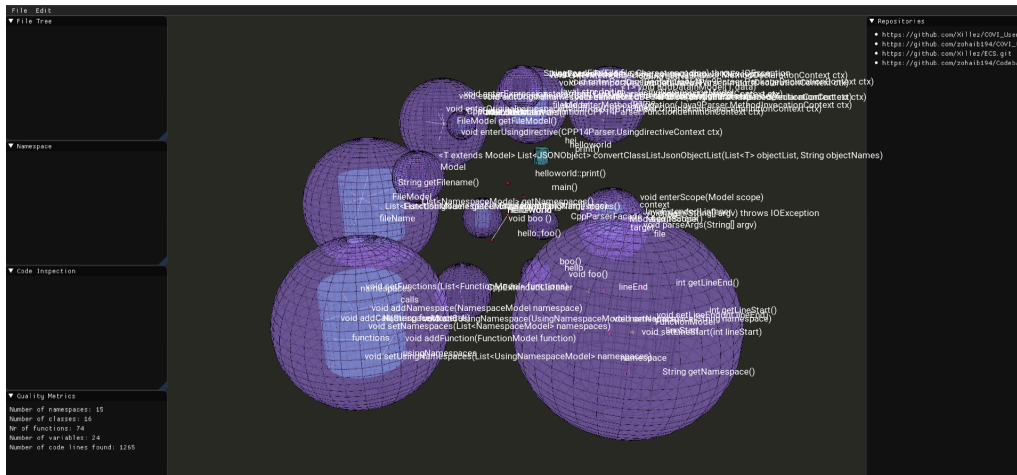


Figure 5: Final UI layout.

The initial [UI](#) concept shown in appendix [E.1](#), was presented to product owner during the second meeting to visualize how the team had intended to design layout of the system. Product owner was pleased by the looks of the system and approved it.

[Tensorflow](#) visualization was an inspiration because it deals with concepts like AI and big data, which have quite understandable and readable ways to represent a big data set through graphs. The team wanted an intuitive way of visualizing the data structure along with keeping the concepts like loosely or highly coupled relationships. Therefore the [Tensorflow](#) representation seemed a perfect fit for this problem.

The left side of figure [E.1](#), was meant to show project related data such as file structure, namespaces, implementation and quality metrics. Whilst the right side shows project independent data, such as repositories registered in the system for easy switching between them.

The final [UI](#) layout is shown in figure [5](#).

5 Development process

5.1 Development methodology

When considering [development methodology](#) it was important to choose one that would fit the project and fit the prioritizations of both the product owner and the development team. The teams project independent priorities were:

- High focus on code
- Embedded documentation
- Self documenting code
- A good fit for teams experience level
- Ability to tackle miss-estimation
- Team focus

Product owner gave the impression that professionalism, high code quality and an independent development team was important.

5.2 Choice of methodology

Apart from the project independent priorities mentioned in the previous section, the team considered these project features as leading when choosing [development methodology](#):

- Ambiguous specification - A general desire was given, but details were left to the development team.
- Research focus - Project differed greatly from other projects.
- Likely to change over time
- Require additional knowledge - The team had limited to no knowledge on some topics as mentioned in section [1.5.1](#)

The team had to plan for high degree of uncertainty, making [agile](#) a requirement. With the teams experience level, it seemed like [Scrum](#) would be a good fit. [Scrum](#) fits well with the focus on code and code-quality, but would still generate enough documentation to help with professionalism. The team used ideas from [DevOps](#) along with [Scrum](#), to improve documentation and embed the documentation together with the code.

5.3 Execution

The team worked with one week long [sprints](#), starting on Tuesday morning and finishing on Monday evening. The [sprint](#) length meant that underestimated tasks would not cause too large of a disruption and the team could quickly change priorities. One [sprint](#) lasted for two weeks as the quantity of work was large and provided little opportunity for change in priorities.

All member worked in the same room during a significant majority of the time, with

high levels of communication. Each member worked with their own tasks taken from the backlog, and requested help if needed.

When a task was submitted, it would be reviewed by both other members and accepted or had changes requested. While reviewing, the team members would often discuss in person with the one who submitted, to assure that both had the same interpretation of the solution. The requirements for a submission to be accepted was quite high, often rejected due to spelling errors or lacking comments. Before a submission it was also required to run [formatter](#), [vetter](#) and [linter](#) in addition to documenting through comments following the relevant [documentation generator](#): [APIDOC](#), [Godoc](#) or [JSDoc](#). Initially the idea was to also do documentation through [IaC](#), but due to the continual underestimation of tasks, it was not followed up on. [Dockerfiles](#) and [Docker Compose](#) are still present and working, but were outdated for a large portion of the development.

zohaib194 added some commits on 20 Feb

- COD-32 Added start line and end line of function body into the initia... 416ad00
- COD-25 Integrated gorilla mux 2fe9173
- COD-25 Setup gorilla mux and setup handler function for GetImplementa... 79443a6

zohaib194 requested review from [eldarht](#) and [Xillez](#) as code owners on 20 Feb

zohaib194 self-assigned this on 20 Feb

Xillez requested changes on 21 Feb [View changes](#)

backend/apiServer/main.go Outdated [Show resolved](#)

COD-25 renamed r to router in main.go 3a98655

eldarht requested changes on 21 Feb [View changes](#)

backend/apiServer/controller/repo.go Outdated [Show resolved](#)

backend/apiServer/model/file.go Outdated [Hide resolved](#)

```
... @@ -0,0 +1,36 @@
1 + //Package model refers to model part of mvc.
```

eldarht on 21 Feb [Collaborator](#) + 🗨️ ...
Should probably be merged with FileModel struct and rest in a struct like CodeSnippetModel or something to show why you got StartLine and EndLine as all "Files" start on line 0 and end on EOF.

zohaib194 on 21 Feb [Author](#) [Owner](#) + 🗨️ ...
Agreed with part that the strcut should be renamed to CodeSnippetModel but i dont agree with that it should be merged with fileModel because the functionality doesn't really require any data from fileModel. The functionality should be part of CodeSnippetModel.

eldarht on 21 Feb [Collaborator](#) + 🗨️ ...
Thats fine, as long as we are consistent.

Figure 6: Pull request discussion.

Project status was documented with [Jira](#), [Confluence](#) and [Git](#).

Mondays were used for meetings and started off by having a meeting with supervisor followed by preparation for meeting with product owner, then said meeting and ended with retrospective meeting or internal meetings.

Meetings with supervisor

Meetings with supervisor were used as a [sprint review meeting](#), discussing what was completed or not in the last [sprint](#), and how the team worked. Suggestions on how to improve the development process or documentation and professionalism were given to the team.

Meetings with product owner

Planning for the meeting with product owner consisted of preparing a suggestion for a [sprint backlog](#). The meetings were used as [sprint review meeting](#) and [sprint planning meeting](#). General recommendations were given as well as specific recommendations on choice of technology or [software design pattern](#). The suggested [sprint backlog](#) was looked over and finalized according to product owner's input.

Internal meetings

Internal meetings were used for [sprint retrospective](#) meetings and for story point estimation of the upcoming [sprint](#).

Story points

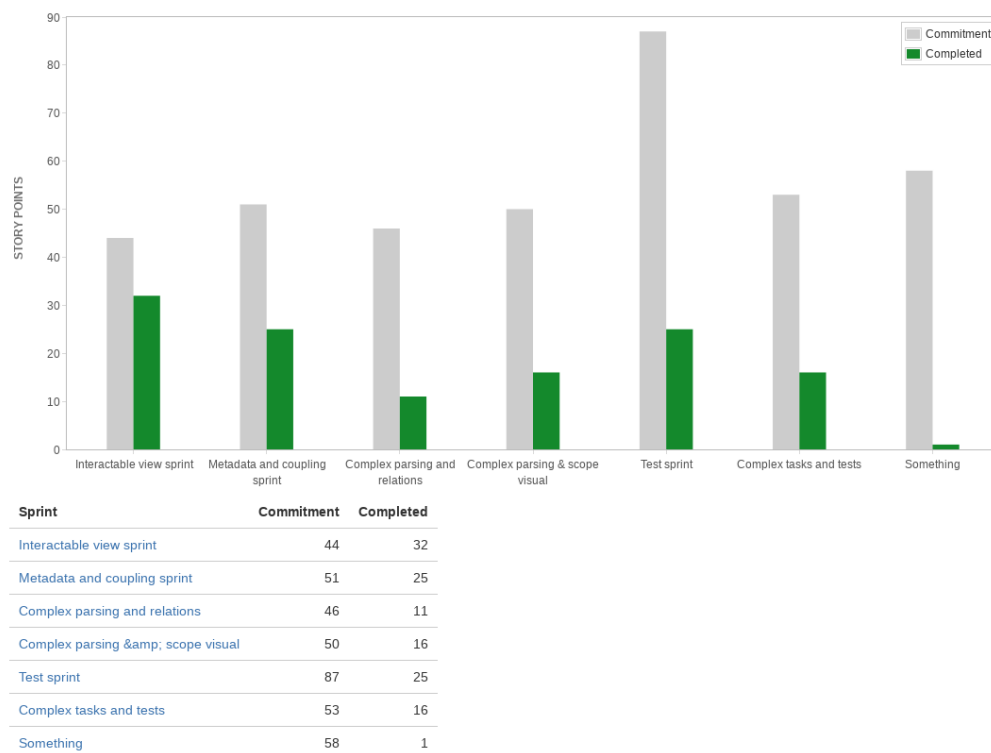


Figure 7: Velocity chart.

For estimation [story points](#) were used. [Story points](#) were given based on relative size, and time value changed over time. This change meant that the teams velocity as shown in figure 7 is misleading.

Scrum board

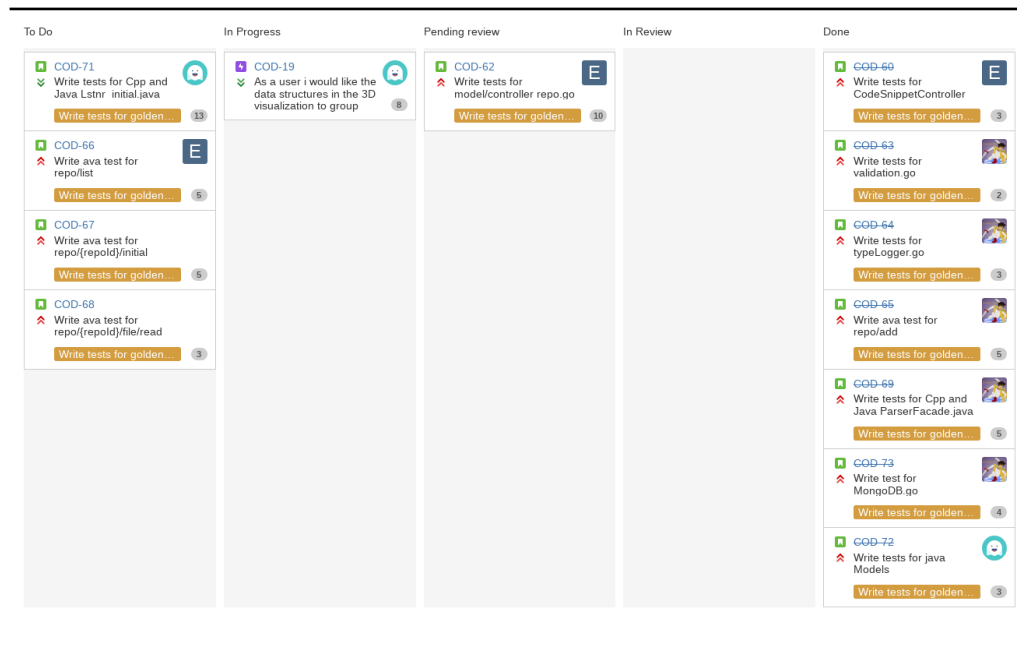


Figure 8: Scrum board for test sprint.

As shown in figure 8, the workflow was divided into five parts:

- "To Do" - [Sprint backlog](#).
- "In Progress" - Tasks currently being worked on.
- "Pending review" - Developer consider the task completed and want other team members to review it.
- "In Review" - Team member has reviewed task and requested changes.
- "Done" - Team members have accepted and merged pull request.

5.4 Sprint Overview

5.4.1 Sprint: hello world (02. feb - 13. feb)

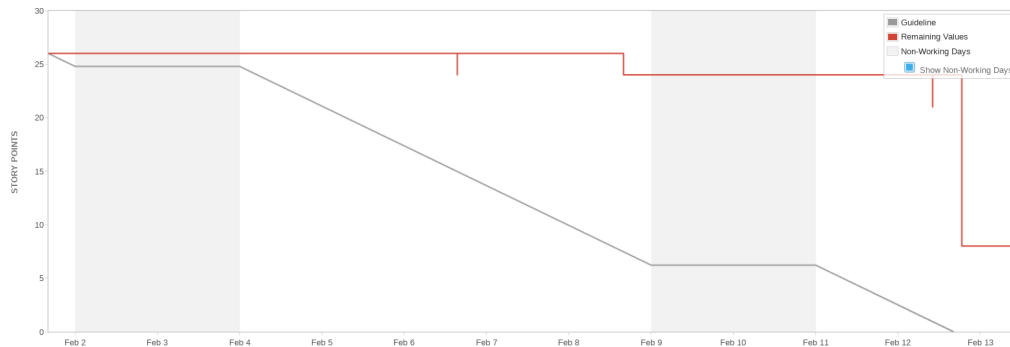


Figure 9: Burndown chart for sprint hello world.

In this sprint the idea was to make a prototype that could represent a function definition from a [Git](#) repository [Uniform Resource Identifier \(URI\)](#). The prototype should also indicate to the user if the browser does not support [WebGL](#).

As figure 9 shows, sprint took more time to finish than planned. The reason was that it took some time to get the parsed code to follow the planned [JSON](#) format. This led to delays in visualizing it in the [front-end](#). [JSON](#) was a blocker for merging, once this was done all remaining task were completed.

Front-end

In [front-end](#), two [HTML](#) pages were partially implemented; home page and visualization page. In the homepage a field for submitting the [Git](#) repository link was created. If a valid repository [URI](#) was submitted, user would be redirected to visualization page where loading would be shown. Once loading was completed a 3D visualization would be presented to the user. There a representation of simple functions with random placement in 3D environment was implemented.

Back-end

In the [back-end](#), [API REST](#) endpoints for initial request and adding [Git](#) repository link to database was implemented, along with java parsing for parts of simple function definitions.

5.4.2 Sprint: representation of complex data (14. feb - 18. feb)

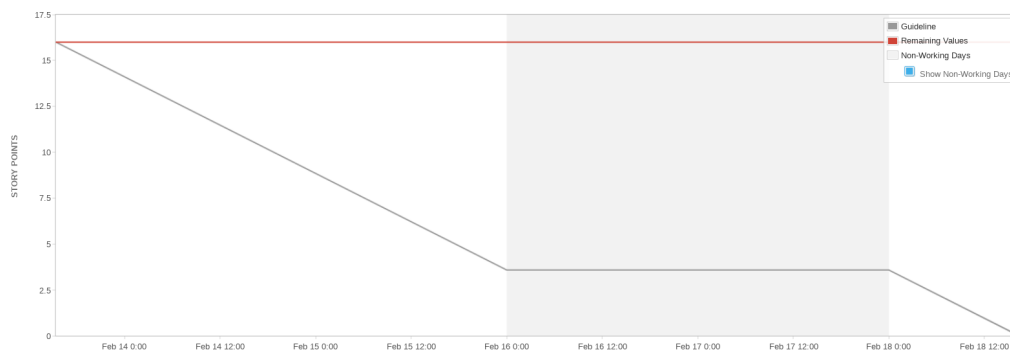


Figure 10: Burndown chart for [sprint](#) representation of complex data.

The goal of the [sprint](#) was to relate the visualization structure to its [source code](#) implementation and group together related structures.

Unfortunately no tasks were fully completed and all got transferred to the next [sprint](#). This is the reason for why the Actual Work Remaining line in Figure 10 doesn't decrease within this [sprint](#). The tasks were underestimated and far too large to be completed.

Front-end

In the [front-end](#), the [Graphical User Interface \(GUI\)](#) for data structure implementation was failed to be integrated. Requirements for fulfilling this task were not satisfied. Therefore updating [back-end](#) was prioritized. A basic version of [FDG](#) that handles parenting relations of data structures was implemented but wasn't considered complete. [Namespaces](#) were also only partially implemented.

Back-end

In the [back-end](#), [JAP](#) was updated for function definition [JSON](#) to include start line and end line for fetching its implementation. [Namespaces](#) were also parsed.

5.4.3 Sprint: interactable (20. feb - 25. feb)

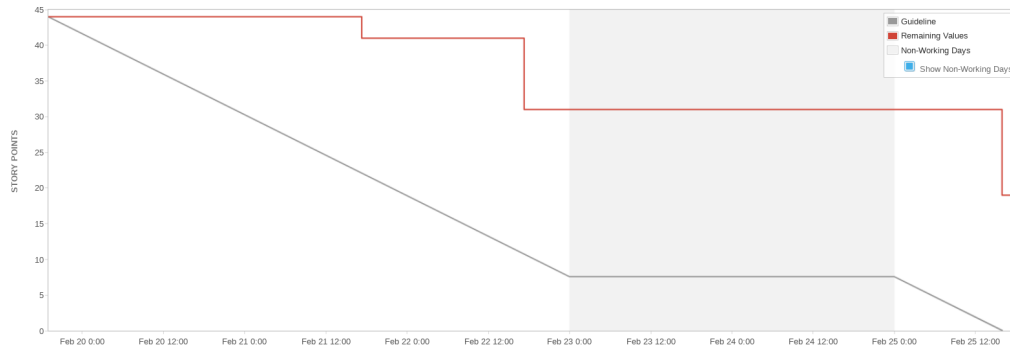


Figure 11: Burndown chart for [sprint](#) interactable.

The goal of this [sprint](#) was to finish previous sprint, along with research on how to use [ANTLR](#) for scope parsing. To improve on the previous [sprint](#), the tasks were divided more finely and declared more clearly.

Front-end

In the [front-end](#), the [GUI](#) layout of the system was implemented on the visualization page. Minor refactoring on [front-end](#) was also done during this [sprint](#).

Back-end

In the [back-end](#), was updated with a new [API](#) endpoint for fetching implementation of functions. For [JAP](#), through research the team found a recommended way to parse scopes from codebase using stack data structure [56]. The task was underestimated and took some extra time to integrate the new implementation in [JAP](#).

5.4.4 Sprint: metadata and coupling (26. feb - 04. mar)

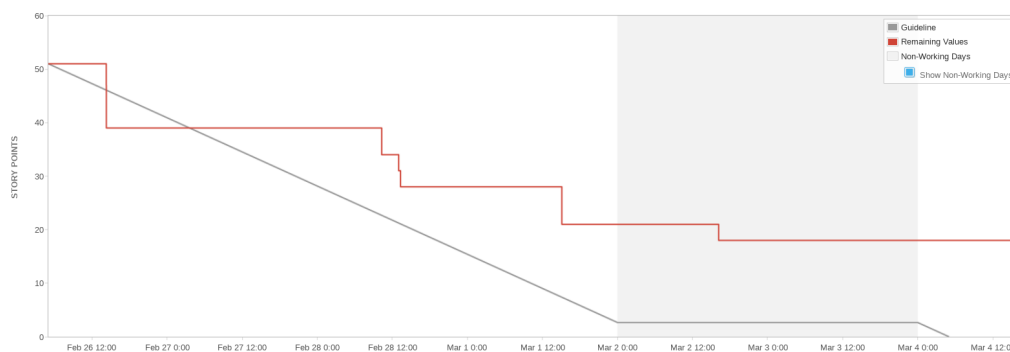


Figure 12: Burndown chart for [sprint](#) metadata and coupling.

The goal of this [sprint](#) was to add data about the project into the representation and visualize function calls.

Figure 12 is a little misleading since it shows that the team was ahead of Ideal Work Remaining line. The reason Actual Work Remaining line dropped was that a task that should not have been in this [sprint](#) was removed. From the previous [sprint](#) the team also noticed that many tasks were [full-stack](#) tasks and took a lot of time to finish. In this [sprint](#) the team made [full-stack](#) tasks into [epics](#), divided them into stories and added sub-tasks under them so that the progress of the [sprint](#) seemed continuous. The team also missed estimating sub-tasks which would have made the [burndown](#) chart look even smoother.

Some of the tasks in this [sprint](#) were still big and depended on other tasks. Tasks such as display function calls was a very difficult [full-stack](#) task that existed throughout all [sprints](#).

Front-end

In the [front-end](#), the team was able to connect the implementation window to show the implementation of simple function once clicked. Quality metrics window was also updated with simple metrics such as [LOC](#), number of functions, classes and [namespaces](#). Repository window was implemented to show the repositories stored in the database, along with a [REST API](#) endpoint to fetch repositories.

Back-end

In the [back-end](#), initial request functionality was updated to use [Websockets](#). This was done due to lack of information on processing of project files for the user on the loading page. Scope sensitive parsing was properly integrated in [JAP](#).

5.4.5 Sprint: complex parsing and relation (05. mar - 11. mar)

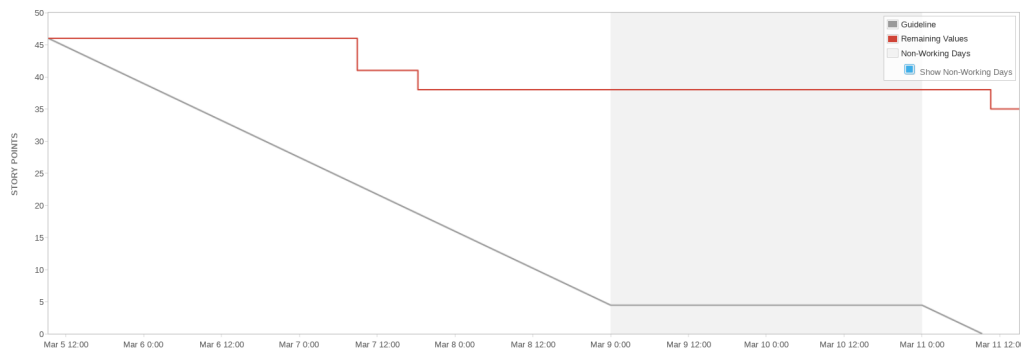


Figure 13: Burndown chart for sprint complex parsing and relation.

The goal of this [sprint](#) was to parse classes and continue improving parsing for function calls, in addition to research on quality metrics and measures.

Front-end

In the [front-end](#), the home page was updated with information on features completeness, browser support and contact information.

Back-end

In the [back-end](#), [Websocket](#) was implemented.

The following were partially implemented or touched briefly due to other tasks having priority, which were too large and underestimated:

- Parsing of classes for visualization.
- Parsing and extension of [API](#) to include function calls for connecting related functions and classes.
- Research on complexity measures and applications/services that could perform these measures.
- View of implementation of functions.

This is also reflected in figure 13.

5.4.6 Sprint: complex parsing and scope visual (12. mar - 18. mar)

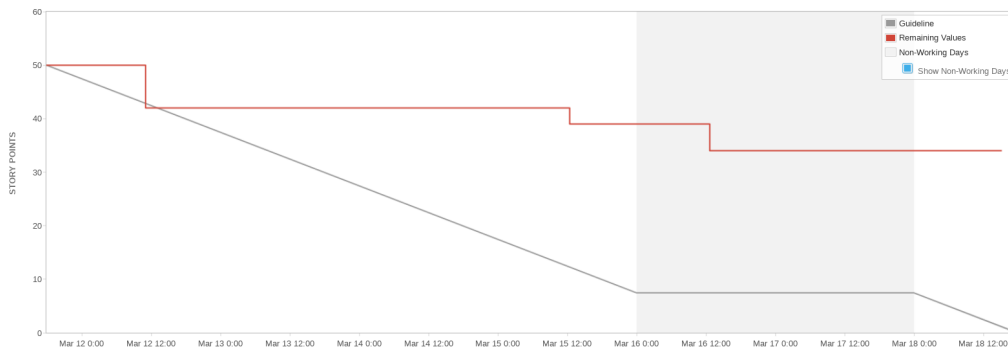


Figure 14: Burndown chart for sprint complex parsing and scope visual.

The goal of this [sprint](#) was to continue parsing function calls and visualize data structures within their respective scope.

As figure 14 shows, some tasks took very long to finish. This was due to underestimation as it was hard to conceptualize all aspects of the task during the [planning poker](#) session. This [sprint](#) was one of the longest [sprints](#) the team went through, in total the team spent over 200 hours. The reason for this was that the team was extensively behind schedule and many of the core functionalities had yet to be figured out.

Front-end

In the [front-end](#), tree structures were used instead of arrays for [FDG](#). Implementation of tree structure was required before visualization of scopes could be done.

Back-end

In the [back-end](#), Go [API](#) endpoint for initial request was updated to give information about function calls. A custom logger was implemented in Go [API](#). Research on different quality metrics such as [Connacense metrics](#), [Cyclomatic complexity](#) and [Halsted complexity](#) was done, along with finding an existing solution that could do static analysis on a specified code block. The team was recommended by supervisor to look into

[SonarQube API](#). This would have worked for our purpose but it would require a lot of configuration and was not done due to prioritization and time limitations. There was a lot of refactoring done in this [sprint](#). In [JAP](#) more listeners provided by [ANTLR](#) were used instead of huge nested if statement blocks. Parsing classes and function calls were partially implemented, but took multiple [sprints](#) to finish due to the flexibility of C++. The reason function calls took too long to finish is explained in more details in appendix [G](#).

5.4.7 Sprint: testing (19. mar - 02. apr)

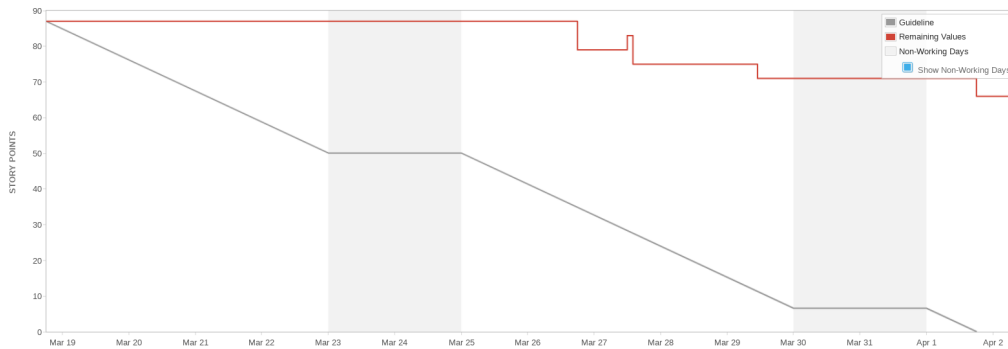


Figure 15: Burndown chart for sprint testing.

This [sprint](#) had a primary focus on testing and since no testing had been done and there were a lot of tests to write. Therefore it was decided to make this a two weeks long [sprint](#).

As the figure [15](#) shows, no task was done in the first week and this is due to no experience with any of the testing framework that were used in this project such as [JUnit](#) and [AVA](#).

Back-end

Tests in [back-end](#) were written for the following files and functionalities.

- CodeSnippetController.go
- validation.go
- typeLogger.go
- repo/add - [API](#)-endpoint
- Cpp and Java ParserFacade.java
- java Models
- MongoDB.go

5.4.8 Sprint: complex task and testing (02. apr - 08. apr)

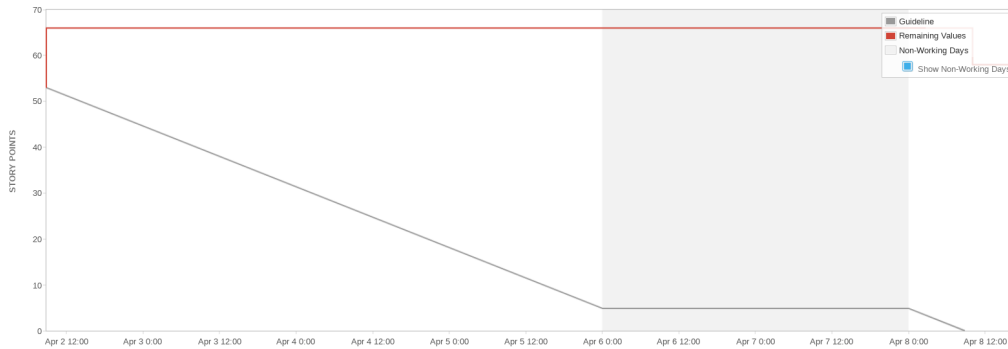


Figure 16: Burndown chart for sprint complex task and testing.

This [sprint](#) was a pair programming [sprint](#) for tasks that were complicated and had existed throughout many [sprints](#). Pair programming was very beneficial to get every group member up to date with the different components of the system that were initially implemented individually. As the figure 16 shows, it still took very long before the team could finish any task. This is due to the tasks in this [sprint](#) being dependent on each other. The tasks were partially done before the team moved over to the dependent task and not finished before the end. This made the pull request so large that the team were not able to accept it before the [sprint](#) was completed. Pair programming helped to get following task done;

- In [back-end](#), the [JAP](#) was updated with a complete implementation of class parsing, along with function calls parsing.
- In [front-end](#), the visualization was updated with scopes inside scopes.

By the end of the [sprint](#) some members felt that their time was better spent on individual tasks rather than pair programming.

6 Implementation

6.1 Front-end web development

6.1.1 Overall design implementation

```

0 var FDG = (function FDG(
1     ...
2 ) {
3     ...
4     var getNodeIndex = function(name) { ... }
5     ...
6     var getNodes = function() { ... }
7     ...
8     // Expose private functions for global use.
9     return {
10         getNodeIndex: getNodeIndex,
11         getNodes: getNodes,
12         ...
13     }
14 }
15 );

```

Listing 6.1: Use of Module pattern in stripped down FDG

The team has a background in [object-oriented programming](#) and very limited experience with JavaScript. Implementation of [front-end](#) started off as simple functionalities but become a problem as the team progressed. [Front-end](#) was responsible for many things such as [FDG](#), graphics, [GUI](#) and etc. It became a problem that the code was expanding and the team did not know how to control files size but also how to properly divide the code sections in different files. During refactoring the team had done research and found different design pattern where module pattern [\[57\]](#) was very similar to classes. "Class" was also found but though research is seemed the module pattern was very commonly used. One reason to choose module pattern was that the team wanted [front-end](#) to be object-oriented, so that only the object would know about its properties and functionalities. Another reason to do so, is that it makes code easily understandable by use of getters and setters, as shown in the figure [6.1](#). The team then decided to implement module pattern to structure front-end.

There were two libraries used in [front-end](#); THREE.js and ImGUI-js. Three.js is a recommended [\[58\]](#) 3D graphics library based on WebGL. It also contains many different functionalities such as scenes, meshes, camera, lights, etc. This fits the requirements and worked for this purpose. Dear ImGUI is a windowing library for C++ that all team members were familiar with from the graphics programming course [\[47\]](#). Dear ImGUI has a binding to JavaScript called ImGUI-js that uses TypeScript and Emscripten. This way the team was able to integrate ImGUI-js into the system. The library is easy to use and fits the teams purpose.

6.1.2 Force-directed graph

The [FDG](#) object maintains a tree of nodes and manages the inter-dependencies between the nodes in the tree. It references the tree through its root, which in turn references its children. The main force calculations and scope handling are done in the nodes, so the [FDG](#) functions as a wrapper.

Position calculation

```

0    ...
1    if (typeof link !== "undefined") { // Attractive forces
2        forceScalar = (
3            link.attraction * Math.log10(
4                dist / (minDistance + size + node.getSize())
5            )
6        );
7    } else { // Repulsive force.
8        forceScalar = (-1 *
9            (
10               (maxDistance + size + node.getSize()) /
11               Math.pow(dist, 2)
12            )
13        );
14    }
15    ...
16    ...
17 gravityForce = (Math.log10(distanceFromOrigin + maxSize) - Math.log10(maxSize -
18    distanceFromOrigin)) * gravityForce;
19    ...

```

Listing 6.2: Force calculation FDG

To calculate the position, [FDG](#) consider the root as the projects global scope, then extracts its children and calculates the force between them. Each child is itself a scope within its parent. The force within each scope is calculated separately to assure that the structure stays within its scope. Listing 6.2 shows that the force calculation depends on whether or not there is a link between the nodes. If there is a link, it is considered a positive attraction, otherwise it is considered a repulsion.

The idea behind the attractive force calculation is to function like a spring; the more energy is required the further apart they are. [Hooke's law](#) was considered for the calculation. The implemented calculation uses \log_{10} on the distance between each node divided by the minimum distance and size of each data structure. This is to create a repulsion when the two structures are too close and a strong attraction when they are too far apart.

The repulsion uses the Inverse-square law [59] to grow weaker over a distance.

The figure also show the gravity force calculation. Gravity attracts all structures towards the center of their scope. The center is the zero vector and updated with parents position when added to the scene graph after all relative positions have been calculated. The gravity calculation is based on [logit](#), as this implementation gives a gravity force approaching infinity when distance between structures approach max distance. This infinite force prevents structures from escaping their parent scope. The problem of limiting the position within a scope seemed related to [sigmoid](#) that the group had ex-

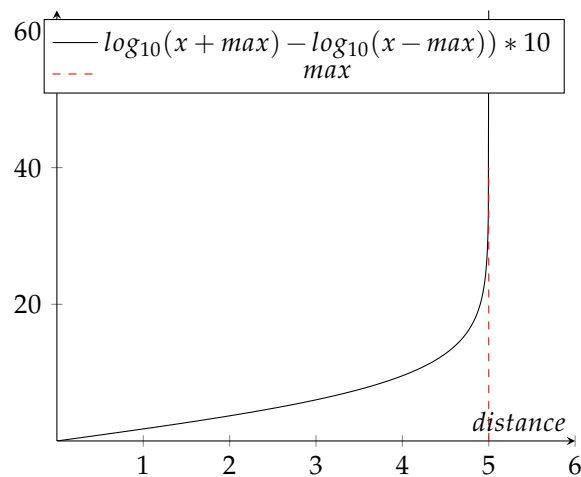


Figure 17: Gravity equation based on logit.

perience with through REA1121 - Mathematics for Programming [49]. [Sigmoid](#) seemed relevant as it allowed for manipulation of its range and gave a smooth curve. Problem with [sigmoid](#) is that if x is distance and y is force, then it would clamp on force rather than distance. [logit](#) was described as the inverse of [sigmoid](#) and would therefore clamp on distance, limiting the structure within its scope. As the distance between center and a structure can not be negative and gravity should never be negative, [logit](#) was altered so it crossed the x axis at origin and approached infinity when it approached max distance like shown in figure 17.

Recursivity of node

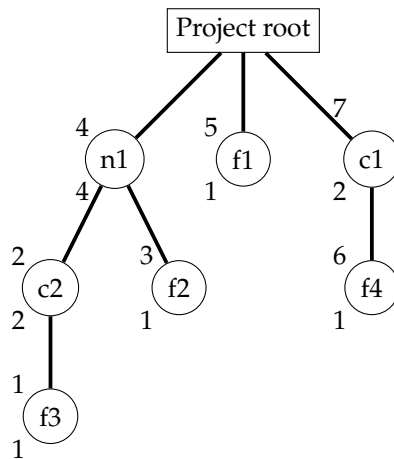


Figure 18: Local to global index example.

Figure 18 shows a project with one namespace, one class and one function in global-scope, one class in the namespace and one function in each class. Top number is global-index and lower number is local-index.

```

0  /**
1  * Gets the node based on index.
2  *
3  * @param {number} requestedIndex - The requested node index
4  * @return {object} The node.
5  */
6  var getNode = function(requestedIndex){
7      var localOffset = 0;                // Used to calculate relative index.
8      var requestedNode = null;
9      // Check if self is requested node.
10     if (typeof index !== "undefined" && (index-1 === requestedIndex)) {
11         return this
12     }
13     // Check if children is requested node.
14     children.every( function(child , i) {
15         localIndex = child.getIndex();
16         // Check if requested is within childs range.
17         if ((localIndex-1)+localOffset < requestedIndex){
18             localOffset += localIndex;
19         } else {
20             // Child contain range with requested node.
21             requestedNode = child.getNode(requestedIndex - localOffset);
22             return false;
23         }
24         return true;
25     });
26     return requestedNode;                // indicate that node was not found
27 }

```

Listing 6.3: Get node function

The [FDG](#) tree requires each node to have an identifier so that they can identify each other for linking and potentially other purposes. This identifier is based on position in the tree and for easy searching. The identifier acts as an index in an array, starting at 0 and increasing with one for each node. When it was implemented, it had to be created based only on information of its sub-tree. This meant its global index in the tree could not be calculated, only its local index where the node is the root of its sub-tree. Once the tree is set in the [FDG](#), all nodes are given their global index. Before the global index is given, it can be calculated from the root node by using the local index given while building the tree. The local index is one greater than the sum of each child's local index. Figure 18 shows the relationship between local and global index while listing 6.3 shows a recursive call to find a node with global index using the local index. LocalOffset in listing 6.3 refers to the offset from global index to local index caused by the parent node having a subtree to the left of the node being checked. Figure 18 is simplified by counting from one instead of zero. The actual global index starts at zero to fit with array index. To account for the local index going from one and global index being accessed from zero, figure 6.3 shows index and local-index being subtracted by one in the conditionals.

6.1.3 Localization

```

0 var LOCALE = (function() {
1   var locale = {
2     sentences: [
3       {
4         id: "language_not_found",
5         values: [
6           {
7             language: "english",
8             value: "Could not find language"
9           },
10          {
11            language: "norwegian",
12            value: "Fant ikke spr\aket"
13          }
14        ]
15      },
16      ...
17    ]
18  };
19
20  // Available languages.
21  const languages = ["english", "norwegian"];
22
23  // English language as default.
24  var currentLanguage = languages[0];
25
26  var getSentence = function(id) { ... }
27  ...
28 })();

```

Listing 6.4: Language option functionality

The current implementation of [localization](#) is not very good due to the way sentences are stored. The reason for this implementation is that it was quick, easy to setup and easy to use. Also the team did not want to spend much time on this feature as it was not a requirement, but would allow for scalability in the future. There might be better ways of doing this, for instance the Mobile Programming course[60] covers the use of Android framework that uses separate files for each locale.

In the current system, sentences are stored as objects within a list, where each sentence has a list of the different languages and the sentence itself in that language. To find a sentence, one first have to find the correct sentence based on the identifier and then find the correct language based on the currently selected language. As more sentences are localized, the time taken to find the latter sentences is increased due to the number of sentences to search through. A better way of implementing this would be to sort primarily on the languages them selves in a [key-value map](#), where key is the language and value is a list of sentences. This would eliminate the use of list for languages and making the search process more effective. To optimize even further, the sentence list inside the [key-value map](#) could also be made into a [key-value map](#) for the same reason.

"getSentence" is the function responsible for fetching sentences in the currently selected language. If the sentence could not be found it defaults to the English language.

6.1.4 Graphics

```

0 var setLinks = function(tree, scene){
1   scene.updateMatrixWorld();
2   var nodes = tree.getSuccessors();
3
4   nodes.forEach(function(node, index) {
5
6     if (node.getDrawableViewIndex() !== -1) {
7
8       node.getLinks().forEach(function(strength, otherIndex) {
9
10        if (nodes[otherIndex].getDrawableViewIndex() !== -1) {
11
12          var material = new THREE.LineBasicMaterial({
13            color: STYLE.getDrawables().link.color
14          });
15
16          var geometry = new THREE.Geometry();
17
18          geometry.vertices.push(
19            new THREE.Vector3().setFromMatrixPosition(drawables.get(
20              node.getType())[node.getDrawableViewIndex()].getMesh().matrixWorld),
21            new THREE.Vector3().setFromMatrixPosition(drawables.get(
22              nodes[otherIndex].getType())[nodes[otherIndex].getDrawableViewIndex()].getMesh(
23                ).matrixWorld)
24          );
25
26          var line = new THREE.Line(geometry, material);
27          scene.add(line);
28        }
29      });
30    }
31  });
32 }

```

Listing 6.5: Linking function for calls

THREE.js uses a scene structure called **scene graph**. The structure in the **scene graph** allows to put object in a hierarchy and if the parent moves in the world, the children follow to retain their position in relation to their parent.

The **scene graph** caused a lot of confusion in linking the different functions based on calls, as the functions were set to a relative position whilst the calls were set using absolute position. The solution for this was to use the **updateMatrixWorld** function from **THREE.js**'s **Object3D** which is the base class of **Scene**. It runs a full update on itself and all its successors within the **scene graph** and updates their world positions. After this, it's possible to use **setFromMatrixPosition** function from **THREE.js**, which uses the world matrix to calculate the actual position from world origin.

Drawing of objects in **THREE.js** can be done by using the **Mesh** object. This object takes a **Geometry** which is the shape made from vertices, and a **Material** which gives the object characteristics and functionalities, e.g. Phong shading [61], shadow mapping [62] capabilities, opacity, etc. These can then be added to the scene and updated if needed.

The generation of the links is performed by looping over each nodes links and fetching the respective node's world position. These positions are given to a **THREE.js Ge-**

ometry and combined with a [LineBasicMaterial](#) object, which results in a line between the two positions.

Within the visualization there are a bunch of shapes. Each shape represents some kind of data structure, e.g. class, function, etc. Instead of building custom shapes, it was decided to use [THREE.js](#) built-in geometries.

The style feature is implemented in the same way as the localization. It contains a [JSON](#) data structure which has a list of color schemes and shape settings. The JavaScript Module pattern was used so the configuration data structure could be encapsulated to maintain a good structure and keep the application as object oriented as possible.

The group wanted the appearance of the application to be customizable to the different vendors, if they have a certain color scheme or style they follow or high contrast styles which highlights different aspects of the program better for those with visual disabilities. To help people that are colorblind it was decided to use different shapes for the separate data structures and a high contrast as well as colorblindness-specific color schemes. These color schemes are not currently implemented, but they will be if any work is continued. The only color schemes added are a SublimeText-inspired scheme and a default color scheme.

6.1.5 Interaction

```

0 /**
1  * On click mouse event function.
2  *
3  * @param      {Event}  event    The event.
4  */
5 function onMouseClick(event) {
6
7     // calculate mouse position in normalized device coordinates
8     // (-1 to +1) for both components
9     mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
10    mouse.y = - (event.clientY / window.innerHeight) * 2 + 1;
11
12    // update the picking ray with the camera and mouse position
13    raycaster.setFromCamera(mouse, camera);
14
15    // calculate objects intersecting the picking ray
16    var intersects = raycaster.intersectObjects(scene.children, true);
17
18    if(intersects !== "undefined" && intersects.length > 0) {
19        var funcName = intersects[0].object.name.substr(0, intersects[0].object.
20            name.indexOf(' '));
21
22        sendGetRequest("http://" + config.serverInfo.api_ip + ":" + config.
23            serverInfo.api_port +
24            "/repo/" + id + "/file/read/?lineStart=" + functionModels.get(
25                funcName).getStartLine() +
26            "&lineEnd=" + functionModels.get(funcName).getEndLine() +
27            "&filePath=" + functionModels.get(funcName).getFileName())
28            .then(json => {
29                windowMgr.setDataStructureImplementation(json.implementation);
30            });
31    }
32 }

```

Listing 6.6: A snippet of raycast onMouseClick

The interaction that exists in the system is interaction with function representations and windows. The text within the window showing data structure implementation, won't wrap but will be hidden and can be seen by temporarily expanding the window through clicking and dragging on the lower right corner of the window. Once released the window will reset to its original layout so that it does not hide the visualization.

The interaction with data structures is done using a technique called [raycast](#). [Raycast](#) is built into [THREE.js](#) which is used for mouse picking objects from a 3D space. The function `onMouseClick` is added as an event listener to JavaScript built in window, which represents window containing [Document Object Model \(DOM\)](#).

As the listing 6.6 shows, first the mouse position is normalized based on screen coordinates, then the raycaster is attached to the camera and normalized mouse coordinates are given so that the ray is cast from the camera to the mouse position, once the user clicks in the scene. The [raycast](#) returns a list of intersected objects. This list is sorted in the order that objects were intersected by the ray. This way the first element of the list will always be object that user wants to interact with.

The interacted object gives its name that was assigned before it was added to the

scene. A [key-value map](#) of function models is created, whilst the project [JSON](#) is being parsed in [front-end](#). This [key-value map](#) contains all metadata about all functions that are visualized and is used to identify the actual model the user has clicked. Once the actual function model is found, it can be used to fetch the models implementation and visualize it in the implementation window.

6.2 Back-end

6.2.1 Overall design implementation

```

0 // API routings
1 util.TypeLogger.Info("%s: Setting up api routes",
    packageName)
2 router.HandleFunc("/repo/add", controller.RepoController{}.
    NewRepoFromURI)
3 router.HandleFunc("/repo/list", controller.RepoController{}.
    GetAllRepos)
4 router.HandleFunc("/repo/{repoId}/initial/", controller.
    RepoController{}.ParseInitial)
5 router.HandleFunc("/repo/{repoId}/file/read/", controller.
    CodeSnippetController{}.GetImplementation)

```

Listing 6.7: Api routes

The [back-end](#) is divided into two parts as mentioned in the Technical Design chapter; Go [API](#) and [JAP](#).

As the listing 6.7 shows, some of the entry points has the repository ID as part of their [Uniform Resource Locator \(URL\)](#). This [URL](#) path variable is sent from [front-end](#) and is used to identify a repository from the database. The placeholder for repository ID is implemented using Gorilla Mux. The idea behind doing this was that the entry point would show what repository the [API](#) should deal with and what operation should be done on it based on the [URL](#).

The `".../initial/"` is the main request that uses [Websocket](#) and communicates with [JAP](#) by giving it a project file at a time, and expects a [JSON](#) with parsed code in return. Whilst being parsed the [front-end](#) will receive a message about the file being processed through "message" channel of [Websocket](#) for server feedback to user. In Go [API](#), parsed code [JSON](#) is added into the project model. Once all project files are parsed the project model is converted into [JSON](#) and sent as a response with the "close" channel of [Websocket](#).

Initially the plan was to use [ANTLR](#) in Go for parsing as [ANTLR](#) has a target for Go. Whilst the team was trying to integrate [ANTLR](#) into Go, it came into focus that the grammar files that the team was planning to use had embedded Java code, which would not have worked in Go. Therefore Java seemed to be natural choice to fallback to for parsing and since [ANTLR](#) was natively made in Java. The team also had some experience in Java through Mobile/Wearable programming course [60]. The package "os/exec" in Go was be used to execute [JAP](#) and receive its output.

There are two libraries used in the [back-end](#): Gorilla mux and JSONObject.

Gorilla mux is a library that is widely used for [URL](#) routing for Go projects. This library was mentioned in the Cloud Technology course [52], so the team already had an understanding of it and made it very easy to integrate.

In [JAP](#) the JSONObject library was used to construct [JSON](#) for each data structure model. Once every line of code has been parsed, the file model which contains many different properties such as namespaces, functions, variables, will loop over each property to store as [JSON](#).

6.2.2 Processing server in Go

REST and WebSocket

```

0 response := WebsocketResponse{
1   StatusText: http.StatusText(http.StatusOK),
2   StatusCode: http.StatusOK,
3   Body: map[string]interface{}{
4     "id":          vars["repoId"],
5     "status":      "Parsing",
6     "currentFile": parserResponse.CurrentFile,
7     "parsedFileCount": parserResponse.ParsedFileCount,
8     "skippedFileCount": parserResponse.SkippedFileCount,
9     "fileCount":    parserResponse.FileCount,
10  },
11 }
```

Listing 6.8: WebSocket success response

Initially the requirements asked for a [REST](#) service, but this was loosened in favor of [WebSocket](#) to allow for updating the user on progress of lengthy tasks. As [HTTP](#) responses has more extensive status information than [WebSocket](#), the team decided to use the responses as a wrapper for a trimmed down [HTTP](#) response with [HTTPStatus](#) code, [HTTPStatus](#) text and body containing the requested information. Listing 6.8 shows a potential response to update user on status of code-base parsing. It sends [HTTPStatus](#) 200 OK, along with what is being parsed, whether it is parsing, done parsing or has failed parsing, how many files have been found and the ratio of files that were parsed or not. Files that are not parsed, might include README.md, [Git](#) files or files with unsupported extensions.

6.2.3 Language parsing in JAVA

```

0 memberdeclaration
1   : attributespecifierseq? declspecifierseq? memberdeclaratorlist? ';'
2   | functiondefinition
3   | usingdeclaration
4   | static_assertdeclaration
5   | templatedeclaration
6   | aliasdeclaration
7   | emptydeclaration
8   ;

```

Listing 6.9: Rule for memberdeclaration

Taken from CPP14.g4 [63] from the ANTLR grammar [Git](#) repository

Listing 6.9 shows the rule defining the syntax for member declaration. It specifies that it can be of 7 forms, where each is separated by a vertical bar or OR gate. The first rule specifies that it might contain an attributespecifierseq rule, followed by what might be a declspecifierseq, in turn followed by what might be a memberdeclaratorlist and ends with the ";" character.

```

0  protected boolean isVariable(CPP14Parser.SimpledeclarationContext ctx){
1      CPP14Parser.InitdeclaratorlistContext initDeclList = null;
2      CPP14Parser.DeclaratorContext declarator = null;
3
4      if(ctx.initdeclaratorlist() != null) {
5          initDeclList = ctx.initdeclaratorlist();
6          if(initDeclList.initdeclarator().declarator() != null) {
7              declarator = initDeclList.initdeclarator().declarator();
8              if(declarator.ptrdeclarator() != null) {
9                  if(declarator.ptrdeclarator().nptrdeclarator() != null) {
10                     if(declarator.ptrdeclarator().nptrdeclarator().declaratorid()
11                        != null) {
12                         return true;
13                     }
14                 }
15             }
16         }
17     }
18     return false;
19 }

```

Listing 6.10: If block example

As the grammar file defines the structure of the program, it was highly desired that as much of the parsing as possible was to be handled by [ANTLR](#). If [ANTLR](#) handled the scoping and definitions then one could be very certain that no data structure were skipped or mishandled. Listing 6.10 shows an example of where checking for variables is done manually, rather than relying on more listeners given by [ANTLR](#). This can create a lot of nested if statements to check what form of the rule or nested rule is being matched. This increased the [Cyclomatic complexity](#) and probability of error while also

relying on the development team knowing all the niche aspects of a particular data structure. The same is true when merging the output of rules by defining an interval by using the start point of a rule and the endpoint of another. If there are optional rules within the context, then the output can vary greatly.

```

0 public void enterFunctiondefinition(CPP14Parser.FunctiondefinitionContext ctx) {
1     ...
2
3     FunctionModel functionModel = new FunctionModel(input.getText(interval));
4     functionModel.setLineStart(ctx.functionbody().start.getLine());
5     functionModel.setLineEnd(ctx.functionbody().stop.getLine());
6     this.enterScope(functionModel);
7 }
8
9 public void exitFunctiondefinition(CPP14Parser.FunctiondefinitionContext ctx) {
10     Model model = this.exitScope();
11
12     if (model instanceof FunctionModel) {
13         this.scopeStack.peek().addDataInModel((FunctionModel) model);
14     } else {
15         this.enterScope(model);
16         ... // Print wrong parent model
17     }
18 }

```

Listing 6.11: Antlr C++ listener example

```

0 public void enterMethodDeclaration(Java9Parser.MethodDeclarationContext ctx) {
1     ... // Get interval were function identifier is written.
2
3     FunctionModel functionModel = new FunctionModel(input.getText(interval),
4         ctx.methodHeader().methodDeclarator().identifier().getText());
5     functionModel.setLineStart(ctx.methodBody().start.getLine());
6     functionModel.setLineEnd(ctx.methodBody().stop.getLine());
7     this.enterScope(functionModel);
8 }
9
10 public void exitMethodDeclaration(Java9Parser.MethodDeclarationContext ctx) {
11     Model model = this.exitScope();
12
13     if (model instanceof FunctionModel) {
14         this.scopeStack.peek().addDataInModel((FunctionModel) model);
15     } else {
16         this.enterScope(model);
17         ... // Print wrong parent model
18     }
19 }

```

Listing 6.12: Antlr Java listener example

Listing 6.11 and 6.12 shows a trimmed down version of the parsing of function definition in C++ and Java respectively. Parsing of the two languages can differ significantly as the naming convention used in the different [ANTLR](#) grammar files differ and C++ also has a more flexible syntax. Java grammar might also use different methods

for defining lists than C++. Due to the differences in Java and C++, they are handled separately based on the file extension.

Java is read by the `JavaParserFacade` and interpreted by `JavaExtendedListener`, while C++ is handled by `CppParserFacade` and `CppExtendedListener` class. Each of the extended listeners extend from their respective [ANTLR](#) listener and are meant to contain common functionality for all types of parsers for their language. Currently only the initial parsers exist. The initial parsers creates the initial data used for the main project visualization.

Both [6.11](#) and [6.12](#) listings show the enter function ending with the `enterScope(functionModel)` call and the exit functions calling `exitScope`. The enter function is called when the parser reaches the beginning of its respective rule while the exit is called at the end. With the `enterScope` being called, all following rules being called would have `functionModel` in memory and know they are matched within the function rule. The check for whether or not the instance of the model is of the expected type in the exit function, were initially undesirable as with a correct and full implementation of the [ANTLR](#) grammar, there would be no possibility for an unexpected model and the check relies on the developers understanding of the data structure.

7 Deployment

7.1 Dockerization

[Docker](#) was used to simplify the system setup for any administrative users and help with documenting the infrastructure. The goal was to be able to launch the system by running the command: "docker-compose up".

With this it would be easier for potential administrators to test the system and figure out if it would be appropriate for their use-case. It would also be easier to continue with [open-source](#) development, as the [Docker](#) instances could be used as a common test bed.

[Docker Compose](#) names four services:

- web - Serves the [HTML](#), [CSS](#) and [JavaScript](#). Mainly deals with the visualization, presenting state and quality metrics.
- api - The Go [API](#) server. Mainly dealing with controlling the database and parser on requests from web.
- mongo_db - The [MongoDB](#) database [17].
- doc - Serves the [APIDOC](#)

The services; web, api and doc each have a defined [dockerfile](#) in the repository, while mongo_db uses mongo:latest, which is the official image.

mongo_db is connected with api but not with the network. This improves the security by not directly exposing the database to the internet.

7.2 Deployment on SkyHigh with HOT

As of the writing of this report, the system is deployed at [SkyHigh](#) and associated with the domain <http://www.covi.tech/>.

It uses [Heat Orchestration Template \[36\] \(HOT\)](#) to automate the server setup process. The template defines three servers:

- development - Meant as a testing server that updates rapidly and might contain bugs.
- operations - Meant to be used by the end users and updated with each milestone where most bugs have been patched.
- master - Meant to run [Jenkins](#) to automate updates of development and operations.

Due to the teams very limited experience with [OpenStack](#) and [Jenkins](#), the servers were rarely updated and down prioritized as it would require more time to learn than what could be allocated.

8 Testing and User Feedback

8.1 Unit testing

me.codvis.ast

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
CppLstnr_Initial		46%		32%	70	98	135	255	11	31	0	1
JavaLstnr_Initial		63%		41%	48	78	86	224	7	29	0	1
Main		0%		0%	14	14	37	37	4	4	1	1
ClassModel		44%		25%	15	20	21	41	7	12	0	1
AccessSpecifierModel		60%		41%	12	17	16	39	6	11	0	1
VariableListModel		45%		n/a	5	10	11	21	5	10	0	1
NamespaceModel		83%		91%	9	31	16	71	7	19	0	1
FunctionModel		80%		100%	8	25	15	62	8	21	0	1
FileModel		85%		91%	7	29	11	66	5	17	0	1
DeclaratorListModel		62%		50%	3	10	6	21	2	9	0	1
FunctionBodyModel		78%		66%	4	12	5	28	1	6	0	1
ScopeModel		57%		n/a	5	7	9	17	5	7	0	1
VariableModel		88%		100%	3	15	6	29	3	13	0	1
CallModel		78%		50%	4	9	6	21	3	8	0	1
UsingNamespaceModel		69%		n/a	3	5	5	13	3	5	0	1
JavaParserFacade		90%		60%	2	6	4	22	0	3	0	1
CppParserFacade		90%		60%	2	6	4	22	0	3	0	1
Model		86%		100%	1	6	3	12	1	4	0	1
CppExtendedListener		100%		n/a	0	4	0	7	0	4	0	1
JavaExtendedListener		100%		n/a	0	4	0	7	0	4	0	1
Total	1,437 of 3,767	61%	196 of 366	46%	215	406	396	1,015	78	220	1	20

Figure 19: Test summary of different Java ANTLR parser files.

Unit testing was done in [JAP](#) using JUnit. In [JAP](#) unit tests were written for most of the models and listeners. The goal of unit testing was to test as many instructions as possible to decrease the probability of errors. Trivial functionalities such as getters and setters were not tested. Each model is tested to see if only data of compatible types would be inserted and would return the correct data once [JSON](#) was requested. Listeners were tested to see if they were able to parse specified context and were able to create correct models that could return correct data within the [JSON](#)-structure. Testing in Java was not complete due to prioritization of other important tasks.

```

PASS
coverage: 90.2% of statements
ok      github.com/zohaib194/CodebaseVisualizer3D/backend/apiServer/util 0
.106s   coverage: 90.2% of statements

PASS
coverage: 30.1% of statements
ok      github.com/zohaib194/CodebaseVisualizer3D/backend/apiServer/model 0
.490s   coverage: 30.1% of statements

PASS
coverage: 15.6% of statements
ok      github.com/zohaib194/CodebaseVisualizer3D/backend/apiServer/controller 0
.133s   coverage: 15.6% of statements

```

Figure 20: Test summary of different Go API files.

In Go [API](#) the goal was to test the [API](#) entry points through [front-end](#) using a [JavaScript](#) testing framework called [AVA](#), while functionality within Go should be tested using Go test. As shown in figure 20, almost all functionalities in the util package are unit tested. This package contains different logging functionalities that are used throughout the Go [API](#). In model package only the database functionalities were tested. In controller package one of the REST endpoints were also tested. This happened before the goal of testing in Go [API](#) was set. The endpoint tested with multiple positive and negative tests cases. To create a general structure in testing code for Go, a [Sublime Text](#) package was used called gotests [64].

```

0 {
1   name: "Valid - new repo - sendAddRequest",
2   websocketURL: "ws://localhost:8080/repo/add",
3   payload: {
4     uri: "https://github.com/example/ECS.git"
5   },
6   wantMessage: {
7     length: 1,
8     messages:
9     [
10      {
11        statuscode: 202,
12        statustext: "Accepted",
13        body : {
14          status: "Cloning"
15        }
16      }
17    ]
18  },
19  wantCloser: {
20    reason: {
21      statuscode: 201,
22      statustext: "Created",
23      body: {
24        status: "Done"
25      }
26    }
27  }
28 }

```

Listing 8.1: JSON structure for a valid test case

In [front-end](#), only one of the endpoints were able to be tested due to time prioritization. The endpoint is implemented as [Websocket](#) and as shown in the listing 8.1, [JSON](#) structure was used as for a test case. This test would add a new repository link and expects onMessage and onClose channel of [Websocket](#) to receive data as shown.

8.2 Code Quality

The team put a lot of effort looking into professionalism and best practises when it comes to code. To maintain high code quality there were a number of steps taken:

- Merge requests in Github were used so any members code would be looked through by at least one other member and errors would be reported and not merged until

resolved. This was also very helpful in the sense of giving all members a peek into the progress and what was done in the areas not touched by them.

- Documentation was prioritized throughout the project, with both commenting for auto-generated documentation but also self-documenting code.
- Conventions and best practises were also prioritized. They are always good to follow because they are tried and tested informal rules to produce easily maintainable and high quality code [65].
- Dependencies, inspiration and ideas were taken from what the team considered professional grade projects, libraries or frameworks. If they are still active in development or maintenance, any recent updates, number of people who contributed, general good structure and open-source were all items the team would check for on any project to determine whether it was created in a professional setting. For example if a project has one contributor or significantly lacking in number of commits for the apparent level of complexity, it would not be considered as a professional grade project unless the code itself was regarded as being good.

8.3 User studies

User studies are very important in the sense that it's a controlled trial of the application. The developers can get an insight into how the users will use the application and give feedback on improvements. This usually results in a higher quality product and the developers can ensure that the application is easy-to-use and stable [66, 67]. As this project is more of a research based project, much of the application is more of a prototype and might not be intuitive. The group therefore performed a user study that is available in Appendix D to get some ideas on how to improve the system.

8.3.1 User categorization

The users of this application are categorized in to three types, Student, Lecturer and Professional:

- **Student** - is anyone studying IT.
- **Lecturer** - is anyone teaching IT.
- **Professional** - is anyone getting paid for developing or researching IT.

The intended result from this classification was to find if the different groups would have different ways of viewing the application. Students might have a more simplistic view and initial understanding of the system and might therefore focus more on usability and basic functionalities. Lecturers might be more analytical and would focus a little more on what the system would give them, how this information was given and why the system would represent the information the way it did. The professionals might view the application from a industrial perspective and might give feedback about IT related industrial standards, quality and more advanced features.

8.3.2 Testing methodology

Before the participant answered the questions in the questionnaire in appendix D, they were instructed to look at a predetermined code-example and asked to draw their representation of the given code-example, as well as some basic tasks that would probe how they use the application. Information like where they looked to improve efficiency of re-

trieving information, mouse movements to see indications on thought process and any control inputs to look for navigability issues and intuitiveness within the visualization were observed.

The primary reasons for selecting these questions was that some would give insights into how the application was used and how easy it was to retrieve any additional information from the home page. Others would probe the visualization itself and give an insight to whether it was intuitive and easy to navigate. Some questions would probe the GUI within the visualization and its layout for retrieving/viewing any information and others would give a general understanding of their role but would not be sufficient to identify the individual.

In a in-formal user study it is important that one does not break any [General Data Protection Regulation \(GDPR\)](#) laws which means no recording of any data that can identify the person testing the application without users consent, the ability for the user to request or delete their data and that is deleted within reasonable time. [55].

8.4 Results

As the user study only involved nine participants, the data was not extensive enough to make any definitive conclusions, the group was however confident that data gathered could still be used to guide future work.

8.4.1 Navigation and UI

The questionnaire indicated that the users felt the navigation was mostly intuitive, but wanted information on how it worked rather than leaving it to guesswork. The activities however, indicated that many of them struggled and did not see it as intuitive, especially moving into the data structures and sideways movement. One thing that seemed clear was that initial camera position, before moving, was too close. This made it difficult for the user to orient themselves. It was also mentioned by a user that information about current orientation would be beneficial.

Several users tried to use the static windows for navigation. This was a feature the team had previously thought about, but had not implemented due to time constraints. It became clear that this should have been prioritized higher.

8.4.2 Visualization and interaction

Most users indicated that the choice of colors and shapes were good, and had little problem identifying the relationship between the visualization and code.

Several users had problems with finding the implementation, whether this had to do with unclear terminology or if changes made by the interaction was hard to spot is unclear. Most users clicked the function to get the implementation as expected, but some did not notice the update in the separate window or noticed but did not draw the connection. An improvement might be to add syntax highlighting, increase the size of the window or highlight to active function in the visualization.

8.4.3 Proposed features

- Information about how to navigate
- Information about orientation

- Customized navigation option
- Save and restore camera position
- Ability to change visuals
- Sidebar navigation
- Position reset button
- Toggle visibility
- Information about call order
- Integrate source code into visualization

Most of the suggestions had been discussed earlier and was planned if time allowed for it, but there seem to be a higher want for enhanced navigation features than expected.

9 Discussion

9.1 Overall reflection and experience

Initially the group wanted to keep very high standards for code quality and professionalism. Although all still strove for this throughout the project the team quickly fell behind schedule, therefore sacrificed on professionalism in order to get working features and ensure scalability. The parts of professionalism that suffered was mostly documentation of the process and unit tests. Part of why the documentation did not stand up to what was suggested in the project plan as shown in the appendix B, was that the project plan template did not fit highly agile workflows like the group followed. The group followed Scrum and had documented sprint retrospective and sprint planning meetings, but the low communication threshold meant that most planning was done verbally and not through writing. It allowed for rapidly changing ideas and refactoring, but very little code that could be considered final and therefore not considered worth writing tests for. This will make it more difficult for potential open-source developers to take over afterwards. Although it is unlikely that others will continue the project it still seemed like a valid compromise.

The low communication threshold meant that those working on relatively complex tasks would ask for a second opinion or ideas for potential solutions. This in turn meant that each group member felt committed and prioritized the more advanced tasks. This was beneficial and helped to solve some of the tasks that could not have been solved by an individual. This also had the negative effects of down prioritizing less complex tasks that could easily give a slight enhancement to the project and that the groups focus could slip. The groups would talk about project unrelated topics more than what was considered optimal and therefore had to spend more time than what was expected. Time log can be found in appendix H.

It was often mentioned that progress was behind schedule set in the project plan, although the schedule comment specifically mentioned that changes were expected and should be embraced. It is good that this pushed for quicker development, but it might also have harmed the scalability of the project and thereby also slowed it down by pushing for features when the plan was not fully formed.

The group was always aware that the project was complicated and required a lot of research and experimentation, but throughout the development it seemed that group members did not expect the same level of completeness. The product owner had at some occasions described the project as a research project, yet the supervisor mentioned that the project requirements should be written as hard requirements, something that conflicted with both the research based and agile nature of the project.

Overall impressions is as expected, the group knew the project was very ambitious and aimed more for an idea than a realistic goal. This meant the learning outcome would be significant and pushed the team to do better than what was realistic. It was hard work, but still very interesting and as a group there was a genuine belief that all mem-

bers would want to continue developing the project after the thesis was finished.

9.2 Use of Jira

Throughout the project the team used [Jira](#) for managing the backlog, sprints and the planning of tasks within these sprints. Some tasks at the start were quite large and required Full-Stack development, e.g. "As a user I would like to see classes parsed and visualized in 3D". These tasks are far too big, especially for the group who has very limited experience with technologies used like [ANTLR](#) and [JavaScript](#). The individual group members would also spend a whole week on the same task and this felt quite daunting after a couple of weeks. The group then decided to make these tasks into epics and segment them down into user-stories and sub-tasks. This made the tasks more finely defined and easier to tackle. This made the group members individually more motivated and made the development process feel less daunting, because a member could complete more tasks in a shorter period of time, therefore gain more satisfaction in seeing multiple tasks getting completed. The break-down of the tasks also had the added benefits in making the burn-down charts look more professional and would more accurately represent the work being done. Despite this, underestimation was still a severe problem and some stories were not estimated due to [Jira](#) treating sub-tasks, stories and epics differently. The [planning poker](#) tool in [Jira](#) worked well for estimating stories, but had problems with subtask. Subtasks did not have an estimation field by default. This might be because [Jira](#) meant for them to be of trivial size. The team did not use them like that, but added an estimation field, however the [planning poker](#) tool did not pick this automatically. In general task estimations were greatly underestimated since it was difficult to see the full length of the task when the task was described orally.

To keep track of the time spent on each task, the group used [Toggl](#) and had it integrated with [Jira](#). This was a slight problem as it required the administrators of our [Jira](#) instance. We also required the administrators for [GitHub](#) integration, project setup and team management. It seemed that [Jira](#), although useful, was not the correct tool for our use case as it made it harder for the team to be independent.

There was also a problem with the [GitHub](#) integration as it did not allow for syncing of the issue tracking. This was a problem because the team wanted to handle the project as an [open-source](#) project and wanted the issues publicly available. An alternative would be to use [GitHub](#) with plug-ins for issue tracking and estimations.

Decisions made during development were recorded using [Confluence](#). The export can be found in appendix [F](#).

9.3 Quality metrics and measures

The quality metrics and measures were originally a very important aspect of the project, but were pushed back due to the complexity and how it required higher completeness of the [ANTLR](#) parsing. The topic was researched and decided to focus on metrics mentioned in sub section [5.4.6](#). For the [Cyclomatic complexity](#), counting the branching factor from conditionals and exit statements seemed like something that could be done and would give a significant benefit. This is due to how it gives good representation of complexity on functions, classes and the project. [Sonarqube API](#) was considered as a way to add this into the project as mentioned in sub section [5.4.6](#), but workload was too

great and this is therefore still in development.

[Connacense metrics](#), dealing with implicit connections within algorithms and between variables, was considered the Holy Grail of helpful statistics, but the complexity and difficulty of calculating this meant that it was decided that the group would not try to add it. The group do not even know if it would be possible to calculate connacence, since it seems like something that would be more like a description in an oral conversation. For instance; the group could not find any reliable way to calculate the algorithmic connacence between a [front-end](#) input validation system and [back-end](#) validation system. If the validation checks if a number is between 0 and 10, then the two validations can be written in different languages, have different names for the validated variable like "i" and "n". The group would have to know the [URL](#) the request is send to, where the back-end redirects to and how the back-end obtains the parameters. The group would have to know that it is a validation layer or that the validation is integrated into the behaviour. Despite this the group still found it useful for describing our own system and discussing potential refactoring. The team considered looking into [taint propagation](#) techniques to get a better understanding of this. The team was familiar with [Taint propagation](#) through the Software security course [54] and knew it was outside the scope of this project.

9.4 IaC

As mentioned in section 5.2, it was initially going to be a greater focus on [DevOps](#). Sadly, due to the amount of work required for each feature and continual underestimation of time required for stories and issues, the team could not finish the [Continuous Integration and Continuous Deployment](#) [35] ([CI/CD](#)) features. [CI/CD](#) were concepts the group really wanted and looked forward to using, but the lacking knowledge of the specific tools needed for this meant that it had to be postponed to focus on development. This was a necessary decision, but still one that was hard to make and one the team did not want to make. The team intended to use [OpenStack](#) with [HOT](#) in such a way that one could automatically tear down and rise virtual servers as was recommended [68]. The team is using [OpenStack](#) and [HOT](#). The resulting topology of the architecture as shown in figure 21 was somewhat unexpected and did not seem good due to the high amount of coupling, but still was fully functional and usable.

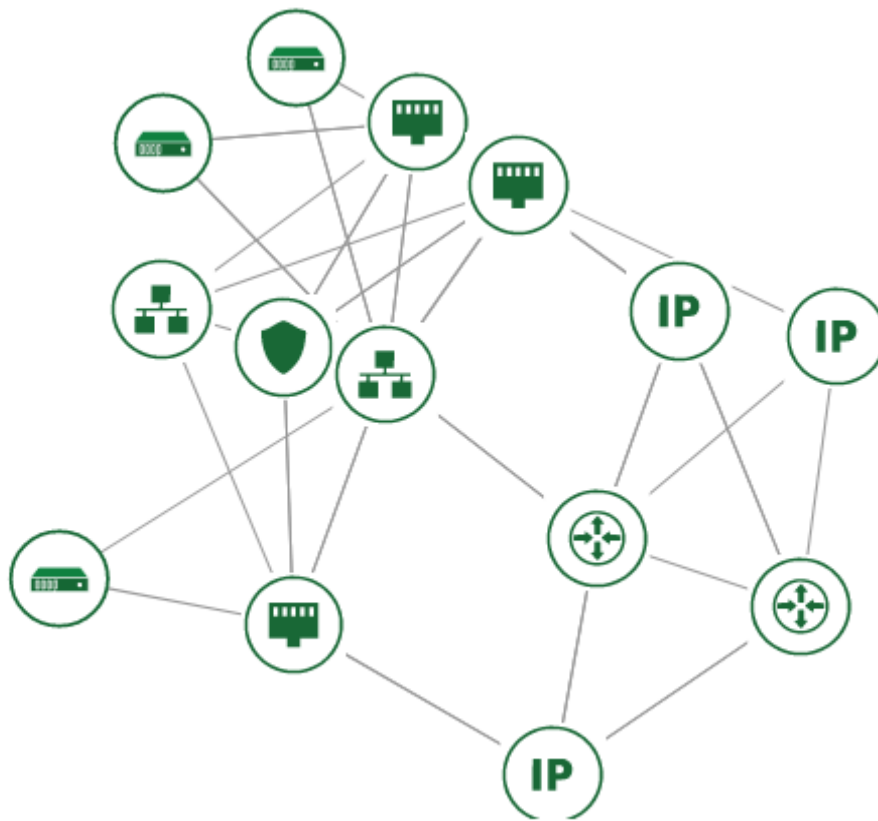


Figure 21: Server network architecture.

The problem with using the stack for an automated process was that the team was unable to easily configure it together with [Jenkins](#) and [Docker](#). The team knew that these services were often used together and with [Puppet](#) or [CHEF](#) but were unable to find what tool was meant for what purpose. In the end the team used the [HOT](#) template tag to run a script for installing docker that would in turn setup [Jenkins](#) master and slave. Then [Jenkins](#) could be configured to handle pushes to [Git](#) repository and update the slaves. As the server configurations were incomplete, it was easier for each group member to host and test on their own local systems instead.

If the server configurations had been completed, the setup could have been used to ensure tests were run continuously and required for pull requests to be accepted.

9.5 Perspective on antlr

9.5.1 Overall reflection and experience

Simple [ANTLR](#) workflow is explained in section 3.1.2. Each listener provides a context that represents a keyword within the file being parsed. The context is a sequential match of tokens. The number of tokens in a rule is the breadth of the context, whilst the depth of the context is the number of nested tokens before you reach a fragment that is a leaf

node and a character sequence. The form of a token is an alternative rule that match the context.

At the beginning of the project the group decided to parse some basic parts of the codebase to get familiar with [ANTLR](#), without thinking much about making it a proper system. This way the group was able to come up with a working prototype in the first week of development, but as the development progressed, the code would not scale due to a few tasks that required huge nested-if blocks to deal with the depth of the context. The problem with if statements is explained in sub section [6.2.3](#).

To retain the professionalism in the code, the group needed to find out a better way to use [ANTLR](#) for code parsing. Through research and experiments the group came up with 3 solutions over time to structure the [ANTLR](#) part of the codebase:

- Stack
 - In the [ANTLR](#) repository it was recommended to use a stack like data structure to process the code [69]. Since [ANTLR](#) parses the code sequentially, a stack could be used to store what is being parsed. This solved our problem with understanding the scope of a context by separating them. As [ANTLR](#) runs on individual files, this approach was unable to identify scopes from different files. This was a problem with the C++ grammar as it sends include statements into a hidden channel. This meant that the group was unable to merge the different files into one project and connect structures and function calls declared in separate files.
- Java Action
 - Java Action was considered to help with contexts having large breadth and many forms where the context is known to have a wanted token. Actions would be for functionality to be added when entering or exiting a context. One could essentially say "When entering this scope; look for all instances of this token". It would return the wanted token no matter what sub-tree it was from, alleviating the need to check what form each nested context has. It would do this by setting an [ANTLR](#) listener to call an Action that is over-written when entering or exiting a scope.
- [ANTLR](#) visitors
 - The [ANTLR](#) visitor were recommended to gain more control in traversing the parse-tree and are usually used if only sub-trees are required to be parsed. The project required many different components of the codebase to be parsed and listeners were therefor the best option for us, where it will go through every node in the parse tree. This way the listeners could sequentially parse every built-in keyword or user defined names from every line on the codebase.

9.5.2 Decreasing cyclomatic complexity in ANTLR

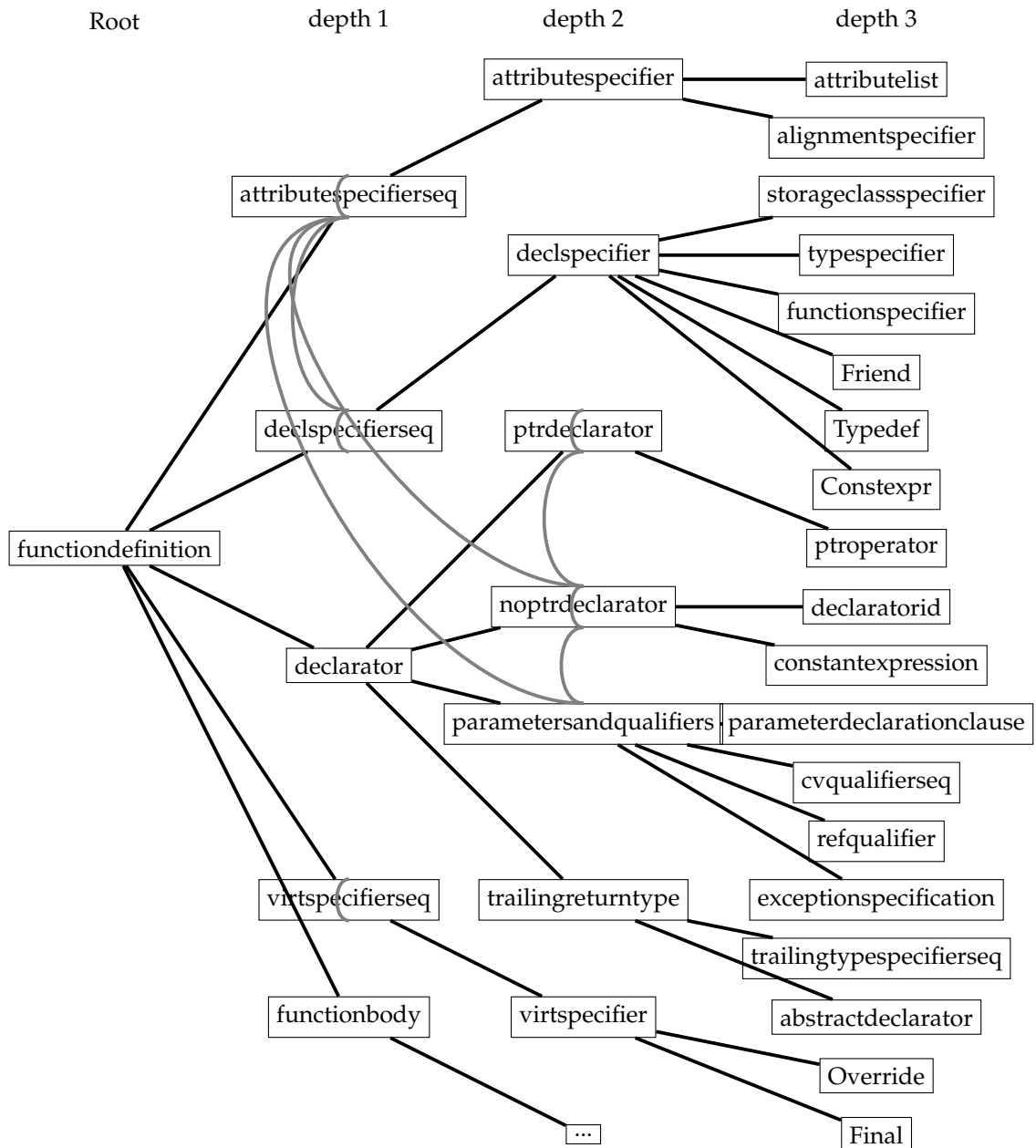


Figure 22: CPP functiondefinition graph to depth 3.

Straight lines show that the, right-hand side is contained within the left side. Curved lines show that something defined earlier or at the same level is also within the rule.

When parsing function definition, it became clear that the depth of the context caused significant complexity as on each level the form of the context had to be checked. This required significantly nested if structures.

The team solved this problem partially using a stack of models. A model represents a structure in the codebase; for instance functions, classes, namespaces, etc. The model

is added to the stack when it's scope is entered and removed after it has completed. This way the team was able to use specific listeners provided by [ANTLR](#) to parse the specific fields for the model.

With the stack, more listeners could be implemented without worrying that a listener would be called at the wrong time and the context be added in the wrong scope. Instead of parsing function definition in depth, one could implement the relevant listeners within the context and they would automatically add their data to their parent model where the parent model was function definition. In figure 22, implementing functiondefinition to enter a function model and implementing typespecifier to add its type to the parent would mean function model would have this without explicitly checking declspecifierseq and declspecifier.

Implementation of listeners in the object-oriented design created a lot of dependency relationships, which increased [connacense metrics](#) of the codebase. As figure 22 shows every node in depth is dependent on its parent node, in most cases some nodes share the same parent node. This made it very difficult to figure out what listener could be executed next. The stack was still considered a good alternative as it removed a lot of [cyclomatic complexity](#) and meant that more of the parsing burden was dealt with, by [ANTLR](#).

Another solution for this could be using [ANTLR](#) visitors, where visitors gives much more control on which token to visit. This way the team could exclude irrelevant subtrees and only visit the information about a specific model. Due to time limitation, visitors were not experimented with, but this could be useful for future refactorings.

9.5.3 Use of regex

Handling function calls was difficult as in Java and C++, the call can be called on an object and that object can in turn be called on any other object recursively. The grammar did not easily allow for the separation of these objects as the languages are very versatile with what an object might be. To bypass the difficulty of using [ANTLR](#) for this purpose, the group decided to use [Regular expression \(regex\)](#). The [regex](#) was used to split the call on "." and "->" that delimited the different objects while other string operations were used to remove the body of some brackets. [Regex](#) could not remove the content of the brackets correctly as they could in turn contain brackets and [regex](#) can not handle [recursion](#). The content of brackets were removed to make it easier to identify the associated function. This approach would not scale in the long run and is likely to create problems. The problems that can arise, come from how this approach prevents [ANTLR](#) from taking care of the recursive structure and niche aspects of the languages that the team has not considered. Removing the body also prevents the system from knowing what overloaded function is being called. Using the [ANTLR](#) built-in features would be preferable in the long term, but would require significantly more work.

The [Regex](#) approach could also be used to parse function pointer assignments as this was a problem due to use of symbol "*" as a pointer and as a multiplication sign. This made it impossible for the team to differentiate between pointer assignments and multiplication expressions. A long term solution to this problem could be updating the grammar file with a rule for pointer assignments, but this was not done since it would require in-depth understanding of the grammar files. Function pointers were dropped

due to time limitations.

9.6 Perspective on REST and WebSocket

As mentioned in sub section 6.2.2, the [REST](#) requirement of the project had to be relaxed. The addition of [Websockets](#) was a necessary decision to take and integrate into the system. [Websocket](#) are built-in to [JavaScript](#), a easy-to-use library for Go made it quick to set up and worked well for the purpose.

[Websocket](#) is a different protocol that uses [HTTP](#) requests to request a [Websocket](#) connection and then switches over to [Websocket](#) protocol if server allows. This allows messages to be sent back and forth over 3 different channels, "open", "close" and "message". The "open" channel is for when the [Websockets](#) opens, "close" is for receiving shutdown message for the connection and "message" is for any information that needs to be sent.

One solution to improve the [REST](#) portion of the [API](#) is to offer [Websockets](#), along with a [REST API](#), which will give next to no information about the parsing progress, but returns the same result.

The use of [Websocket](#) in Go led to very huge functions and this resulted in decreased readability of the endpoints. Huge functions were the side effect of error handling of [Websocket](#). To improve readability the team could break the the functionality of the endpoint into multiple small functions.

One mitigation for the massive amount of error handling would be to use a concept called middleware. This is when you have code that is executed before or after any given code snippet for a specific reason. For example one could wrap a log-in functionality in middleware that would write to a log-file. This could log the users attempt and subsequent success or failure of the log-in. As well as logging, middleware could be used for prepping or extracting input to sensitive functions ensuring no [taint propagation](#) and injections, as well as handling errors.

9.7 Perspective on Graphics systems

9.7.1 Overall reflection and experience

Selection of a graphics system was relatively easy. The whole group had experience with OpenGL from the Graphics programming course [47] and [WebGL](#) is closely related. Everyone already knew the details on how to draw in 3D, but since it was a new language the group didn't have much experience in, the group setup a workshop to familiarize with [JavaScript](#) and [WebGL](#). The group also found THREE.js which is a well-known and well-used graphics framework in [JavaScript](#). THREE.js handles [WebGL](#) automatically, was easy to use, contains ready made shapes called geometries and have a lot of support functionality like vector classes with built-in math. The group therefore decided to use THREE.js.

The project required to show information and interactable [UI](#), for this reason the group researched different [UI](#) Libraries/frameworks found PixiJS, TWO.js, dat.GUI and ImGUI-js.

9.7.2 Choice of graphics framework

Using THREE.js meant that the graphics handling was not a problem, but there were still a problem with using the parsed codebase to create the 3D representation. This was handled in [JavaScript](#), although later it seemed like parts of it could be handled by the [back-end](#) and allow for an improved web [API](#) and better caching opportunities.

9.7.3 Choice of UI framework

The group thought about creating a custom [UI](#) system using PixiJS or TWO.js instead of using other existing libraries. This seemed like a lot of management and adjustment work, considering it would only be used for the visualization.

dat.GUI is a pure [JavaScript](#) and HTML [UI](#) Library. Although it seems easy to use and could possibly work, it seemed too simple from available [examples](#). This would require large amounts of configuration code and would stretch the library's capabilities, which wasn't really desired. Only scenarios where "dat.GUI" would fit well is for a settings or configurations drop-down menu.

ImGUI-js is a 3rd party library and must work in unison with ImGUI, but the difference from TWO.js and PixiJS is that ImGUI-js wraps ImGUI, so in-effect they're one library and this made it very difficult to integrate, therefore lot of time went into making it work properly, but the group did not have to get two libraries to work at the same time. Another benefit of ImGUI is that the [UI](#)-elements exists inside windows which are by default movable and scalable. This results into a very customizable [UI](#) layout out of the box if this is wanted.

9.7.4 Choice of Application Lifecycle Framework

The group considered to use framework for [front-end](#) development because the use of framework is often recommended, especially when using [JavaScript](#) due to the way it handles scopes. The supervisor recommended multiple frameworks such as POLYMER.js, VUE.js, AngularJS and REACT.js. The project is a web-based application which has only two HTML pages and most of the [front-end](#) deals with graphics.

Considering this the team later realized that there was no need for it. Most [Application Lifecycle Frameworks \(ALFs\)](#) are not designed for graphics and therefore did not fit well with this application. It could restrict and require time to learn without giving any significant benefits.

9.7.5 Use of Force Directed Graph library

There are a multitude of libraries out there that handles [FDG](#) quite good like [3d-force-graph](#) and [Springy.js](#), but none of these libraries allowed for individual link strengths. They only have a single global setting for all attractive links, and the reason this was wanted was due to the different data structures being more or less attracted to other certain data structures. Therefore it was decided to not not integrate any [FDG](#) library.

9.7.6 Graphics system refactoring

The 3D representation was changed multiple times but not fully refactored so it still contains code that should have been deprecated, removed or changed to fit the rest of the system. Some of these changes were made late in the project when time-constraints were causing problems. The time-constraints meant that proper refactoring would not

improve the velocity of the team and were therefore not prioritized.

9.8 Perspective on FDG

9.8.1 Overall reflection and experience

The codebase visualization had to be intuitive and able to handle both large and small datasets without any manual adjustment by the user or administrator. A [FDG](#) seemed like a natural choice for automatically adjust the layout based on the scale and composition of the codebase. It also seemed natural to use the scoping in object oriented programming to hide or reveal details based on what the user was looking at.

As this was the first time the group had used [FDG](#), it was a surprise how flexible the literature was about the implementation specification. The literature helped establish a few terms:

- Node - A representation of codebase structure with positional data and metadata required by [FDG](#) to update position.
- Energy - relating to the total energy in the system and what the [FDG](#)'s goal was to minimize.
- Attractiveness - metadata of node relating to how much energy is required to keep it apart from one other node.
- Repulsiveness - metadata of node relating to how much energy is required to keep it close to one other node.
- Links - metadata of node, a map of all repulsive and attractive forces to all other nodes.
- Gravity - A global force affecting all nodes, pulling them towards a defined center.

9.8.2 Detailed experience on refactoring of FDG

The [FDG](#) is probably the part in the program that caused the most problem when trying to expand on it and refactor it. It started off as a prototype with limitations:

- Assumed no nested structures as output.
- Assumed a global center that all structures should gravitate towards to keep the graph centered.
- Assumed an initial repulsion from one structure to another, that's not related and of equal strength.
- Assumed the repulsive forces was negative by the algorithm.
- Assumed all repulsive forces were set as -1 by the parsing.
- Assumed a connection or attraction from structures based on nested structures in input.

The [FDG](#) used Inverse-square-law to calculate force based on repulsiveness and Hooke's law to calculate a force from attractiveness. All the nodes were parsed from the initial nested codebase structure in pre-order and stored in a single array within the [FDG](#) object. The initial structure allowed for relatively simple parsing and well understood and fairly versatile structures. Initially it was desirable for nodes to contain minimal information and only the information required by [FDG](#). This would improve the object oriented model and minimize the amount of data being passed around. The desire to keep nodes minimal was lessened when the group looked into [JavaScript](#) as a

"Share by Calling" language.

When it came to changing [FDG](#) to allow for nested structures as output, it soon became clear that large portions would have to be changed. The wish was to have a system that would be streamlined and versatile. This meant to change array containing all the nodes into a generic tree that more closely represented both the input and output data. It was required to change the single array approach as allowing for nested output, which meant the [FDG](#) would have to run on each nested part separately. Changing the array proved complicated, as links were added while parsing the input and used the index in the array to identify the other node, with a tree, the index would have to be calculated and kept constant. It was thought that keeping the index constant could only be done by parsing the input level-order or pre-order, and this would not allow for nested output as the size of each node was dependent on its children. Therefore the parsing was done post-order and contained a local index that could be used to calculate the global index in the tree when connected to the root. This result is described in sub section [6.1.2](#). This assumption turned out to be wrong, and giving the nodes a global index when building the tree would have been beneficial.

The local to global calculations caused a lot of confusion and significantly increased complexity. The new system had the same cyclomatic complexity as before, keeping conditionals essentially the same, but increased the use of complicated [recursion](#) and had a high connacance with the removed array through the global index.

Although there were made mistakes during the refactoring and it increased the complexity, it was still beneficial. Benefits of using the tree structure were evident, as the group could run [FDG](#) on each node separately from the world origin and input this to the [scene graph](#) to handle offsets and potential navigation. From parsing in the [back-end](#) to the visualization in [front-end](#), the core data is always handled as a tree structure. This means there is a high level of consistency throughout the system but also an implicit connacance that means if the structure is changed in one part of the system then all other parts will also have to be changed.

9.9 "sloccount" vs "Kloc" vs "wc -l"

One of the simplest complexity metrics to add was [LOC](#) and was added as a first, simple metric. Sloccount, [KLOC](#) and wc are tools that can get the [LOC](#). wc was chosen. wc is a linux command line tool to read words, newlines and bytes and prints out the result to standard output channel. wc is language independent meaning it can take any language file as an argument. It can also read only the specified lines within the files by giving the "-l" flag when executing. This worked well for our purpose and therefore used to fetch the implementation from source code files. The other alternatives were considered because wc would not be able to distinguish between comments, code or empty lines, but the other alternatives had the same limitations or would not support enough languages for our purpose.

9.10 Platform for the product

The reason of choosing web as the platform for the the system was for it to be cross-platform and so that any user could easily access the system from anywhere. Also the team did not have much experience in the web-technology therefore it was a good op-

portunity to implement the system as web-service. [JavaScript](#) is the technology the team didn't not have much experience with and become a problem to scale as responsibilities of [front-end](#) increased over time.

9.11 Post-phoning/removing userstory allowing user to submit lexer files

There was an idea early on that the user would be able to input a lexer file for any language and be able to use it to visualize projects written with-in that language. It became clear very quickly the more the team worked with [ANTLR](#) that this feature was not going to be integrated. The listener files are generated based on the grammar file and are language specific, which means one would have to implement each listener independently for each language for the system to be able to parse that language. So to integrate the submitted lexer files, the application have to either inject the new code in during runtime which is far beyond the project scope. It was decided to drop this functionality due to the shear size of this feature, which on its own can be counted as a bachelor if not even more.

10 Conclusion

10.1 Result

The result is a prototype capable of handling Java and C++ and gives a decent indication of simple code quality and complexity. It visualizes data structures in the 3D environment in a clear and intuitive manner, although contains some duplicate data. The product owner requested a system that could be used to identify good or bad libraries, which the system gave a minimal but sufficient indication of through the visualization. It provides a good bedrock for future development, graduation projects or master thesis.

10.2 Future Work

The project could be continued in a master thesis as a research or personal project where some of the following items could be integrated or improved upon:

- Integrate AI to do language recognition.
- Integrate virtual reality for visualization.
- Extend a [REST API](#) to be more independent from systems [GUI](#), but also fetch desired parts of parsed code through [REST](#).
- Spacing as sizing algorithm could be implemented in a proper manner insuring children do not escape their parents nor would collide with siblings.
- File tree structure could be implemented and visualized within the window.
- [front-end](#) could be refactored to a properly segmented and professional system.
- The camera controls could be improved to become more intuitive
- Adding user settings/account capabilities for remembering locale, submitted repositories, custom styles and saving the world state properties to continue the inspection from the same point or preparation.
- Integrate GraphQL to fetch specific scopes from the parsed code, to save on parsing and loading resources.

10.3 Evaluation of group work

Throughout the development, all team members worked well and tried their best to create the best product possible. Despite problematic tasks and new technologies to learn, the group progressed and overcame most hinders.

10.4 Ending

All in all the group is happy about the work done in this project and the knowledge acquired throughout. The group is happy about the quality and consider the project well made, although a few parts could have been done better.

Bibliography

- [1] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. 2001. Manifesto for agile software development.
- [2] Foundation, A. S. Welcome! - the apache http server project. Accessed: 2019-04-03. URL: <https://httpd.apache.org/>.
- [3] Rottmann, P. apidoc. [Online; accessed 16-May-2019]. URL: <https://github.com/apidoc/apidoc>.
- [4] Overby, S. 7 devops roles you need to succeed. URL: <https://techbeacon.com/devops/7-devops-roles-you-need-succeed>.
- [5] Wubben, M., Sorhus, S., & Demedes, V. Ava. [Online; accessed 5-May-2019]. URL: <https://github.com/avajs>.
- [6] of the English Language, A. H. D. 2011. back-end. Retrieved May 16 2019. URL: [fromhttps://www.thefreedictionary.com/back-end](https://www.thefreedictionary.com/back-end).
- [7] Wikipedia contributors. 2018. Burn down chart — Wikipedia, the free encyclopedia. [Online; accessed 4-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Burn_down_chart&oldid=868816075.
- [8] Chef. Continuous automation for the continuous enterprise. [Online; accessed 11-May-2019]. URL: <https://www.chef.io/why-chef/>.
- [9] Wikipedia contributors. 2019. Control flow — Wikipedia, the free encyclopedia. [Online; accessed 3-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Control_flow&oldid=885162294.
- [10] Wikipedia contributors. 2019. Coupling (computer programming) — Wikipedia, the free encyclopedia. [Online; accessed 5-May-2019]. URL: [https://en.wikipedia.org/w/index.php?title=Coupling_\(computer_programming\)&oldid=890889110](https://en.wikipedia.org/w/index.php?title=Coupling_(computer_programming)&oldid=890889110).
- [11] Max Rehkopf. 2018. Epics, stories, themes, and initiatives. [Online; accessed 4-May-2019]. URL: <https://www.atlassian.com/agile/project-management/epics-stories-themes>.
- [12] Hooke, R. 1678. *Lectures de potentia restitutiva, or of spring explainin the power of springing bodies*. John Martyn.
- [13] Jenkins. Jenkins user documentation. [Online; accessed 9-May-2019]. URL: <https://jenkins.io/doc/>.

-
- [14] Beck, K., Gamma, E., Saff, D., & Clark, M. Junit - about. [Online; accessed 5-May-2019]. URL: <https://junit.org/junit4/>.
- [15] Len. Jan 2011. Sigmoid and logit function. URL: <https://blacklen.wordpress.com/2011/01/01/sigmoid-and-logit-function/>.
- [16] Black, P. E. 11 February 2019. Manhattan distance. [online] Accessed:16-May-2019. URL: <https://www.nist.gov/dads/HTML/manhattanDistance.html>.
- [17] Inc, M. What is mongodb? [Online; accessed 8-May-2019]. URL: <https://www.mongodb.com/what-is-mongodb>.
- [18] Wikipedia contributors. 2019. Open source — Wikipedia, the free encyclopedia. [Online; accessed 3-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Open_source&oldid=894916949.
- [19] Foundation, O. Welcome to openstack documentation. [Online; accessed 8-May-2019]. URL: <https://docs.openstack.org/stein/>.
- [20] Wikipedia contributors. 2019. Planning poker — Wikipedia, the free encyclopedia. [Online; accessed 5-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Planning_poker&oldid=888556553.
- [21] Ken Schwaber, J. S. nov 2017. The scrum guide. Accessed: 2019-04-03. URL: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100>.
- [22] Puppet. How puppet works. [Online; accessed 11-May-2019]. URL: <https://puppet.com/products/how-puppet-works>.
- [23] of Engineering, M.-H. C. E. Software metric. Accessed:2019-04-03. URL: <https://encyclopedia2.thefreedictionary.com/software+metric>.
- [24] Wikipedia contributors. 2019. Ray casting — Wikipedia, the free encyclopedia. [Online; accessed 5-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Ray_casting&id=893943246.
- [25] Wikipedia contributors. 2019. Regular expression — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Regular_expression&oldid=895954942. [Online; accessed 11-May-2019].
- [26] Döllner, J. & Hinrichs, K. 2000. A generalized scene graph.
- [27] 2008. About sonarqube. URL: <https://www.sonarqube.org/about/>.
- [28] Wikipedia contributors. 2019. Source code — Wikipedia, the free encyclopedia. [Online; accessed 4-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Source_code&oldid=894961033.
- [29] Skinner, J. & Bond, W. Sublime text - a sophisticated text editor for code, markup and prose. [online], Accessed:17-May-2019. URL: <https://www.sublimetext.com/>.

- [30] Chess, B. 2007. *Secure programming with static analysis*. Addison-Wesley, Upper Saddle River, NJ.
- [31] tensorflow. Tensorflow. [online], Accessed:16-May-2019. URL: <https://www.tensorflow.org/>.
- [32] Wikipedia contributors. 2019. Toggl — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Toggl&oldid=884068908>. [Online; accessed 11-May-2019].
- [33] Wikipedia contributors. 2019. WebGL — Wikipedia, the free encyclopedia. [Online; accessed 4-May-2019]. URL: <https://en.wikipedia.org/w/index.php?title=WebGL&oldid=892586864>.
- [34] Encyclopedia(1981-2019), C. D. white box testing. Accessed:2019-04-03. URL: <https://encyclopedia2.thefreedictionary.com/white+box+testing>.
- [35] Azeri, I. Jan 2019. What is ci/cd? URL: <https://www.mabl.com/blog/what-is-cicd>.
- [36] Foundation, O. Heat orchestration template (hot) specification. [Online; accessed 8-May-2019]. URL: https://docs.openstack.org/heat/latest/template_guide/hot_spec.html#hot-spec.
- [37] Khaloo, P., Maghoumi, M., II, E. M. T., Bettner, D., & Jr., J. J. L. 2017. Code park: A new 3d code visualization tool. *CoRR*, abs/1708.02174. URL: <http://arxiv.org/abs/1708.02174>, [arXiv:1708.02174](https://arxiv.org/abs/1708.02174).
- [38] Wettel, R. *Software systems as cities*. PhD thesis, Università della Svizzera italiana, 2010.
- [39] Wikipedia contributors. 2019. Software development — Wikipedia, the free encyclopedia. [Online; accessed 3-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Software_development&oldid=893601234.
- [40] Krueger, C. W. June 1992. Software reuse. *ACM Comput. Surv.*, 24(2), 131–183. URL: <http://doi.acm.org/10.1145/130844.130856>, [doi:10.1145/130844.130856](https://doi.org/10.1145/130844.130856).
- [41] Mileva, Y. M., Dallmeier, V., Burger, M., & Zeller, A. 2009. Mining trends of library usage. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, 57–62. ACM.
- [42] Nguyen-Duc, A. 2017. The impact of software complexity on cost and quality - A comparative analysis between open source and proprietary software. *CoRR*, abs/1712.00675. URL: <http://arxiv.org/abs/1712.00675>, [arXiv:1712.00675](https://arxiv.org/abs/1712.00675).
- [43] Spinellis, D. 2003. Reading, writing, and code. *Queue*, 1(7), 84.
- [44] Westland, J. C. 2002. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62(1), 1–9.

- [45] Hjelmås, E., of Information Security, D., & Technology, C. Course - operating systems imt2282. Accessed online: 16-May-2019. URL: <https://www.ntnu.edu/studies/courses/IMT2282#tab=omEmnet>.
- [46] Røise, T. & of Computer Science, D. Course - software engineering - imt2243 - ntnu. Accessed online: 16-May-2019. URL: <https://www.ntnu.edu/studies/courses/IMT2243#tab=omEmnet>.
- [47] McCallum, S. & for datateknologi og informatikk, I. Emne - grafikkprogrammering - imt2531. URL: <https://www.ntnu.no/studier/emner/IMT2531/2017#tab=omEmnet>.
- [48] McCallum, S. & of Computer Science, D. Course - game programming - imt3601. Accessed online: 16-May-2019. URL: <https://www.ntnu.edu/studies/courses/IMT3601/2016#tab=omEmnet>.
- [49] Jensen, B. T., of Information Security, D., & Technology, C. Course - mathematics for programming - rea1121. URL: <https://www.ntnu.edu/studies/courses/REA1121/2016#tab=omEmnet>.
- [50] Jensen, B. T. & of Mathematical Sciences, D. Course - mathematics for computer science - rea1101. URL: <https://www.ntnu.edu/studies/courses/REA1101#tab=omEmnet>.
- [51] Haug, F. & of Computer Science, D. Course - algorithmic methods - imt2021. URL: <https://www.ntnu.edu/studies/courses/IMT2021#tab=omEmnet>.
- [52] Frantz, C. K., Nowostawski, M., & of Computer Science, D. Course - cloud technologies - imt2681. URL: <https://www.ntnu.edu/studies/courses/IMT2681/2018/1#tab=omEmnet>.
- [53] Giannakos, M. & of Computer Science, D. Course - web technologies - it2805. URL: <https://www.ntnu.edu/studies/courses/IT2805#tab=omEmnet>.
- [54] Katt, B., of Information Security, D., & Technology, C. Course - software security - imt3501. URL: <https://www.ntnu.edu/studies/courses/IMT3501/2018#tab=omEmnet>.
- [55] UNION, E. O. R. F. D. E. 2016. Lov om behandling av personopplysninger (personopplysningsloven) europaparlaments- og rådsforordning (eu) 2016/679 av 27. april 2016 om vern av fysiske personer i forbindelse med behandling av personopplysninger og om fri utveksling av slike opplysninger samt om oppheving av direktiv 95/46/ef (generell personvernforordning) [gdpr]. https://lovdata.no/dokument/NL/lov/2018-06-15-38/KAPITTEL_gdpr#KAPITTEL_gdpr.
- [56] Parr, T. 2013. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.
- [57] Patel, D. Jul 2016. 4 javascript design patterns you should know. URL: <https://scotch.io/bar-talk/4-javascript-design-patterns-you-should-know>.

-
- [58] Jan 2018. Why to use threejs in web application development? URL: <https://www.cmarix.com/why-to-use-threejs-in-web-application-development/>.
- [59] of the English Language, A. H. D. inverse-square law. Retrieved May 17 2019. URL: <https://www.thefreedictionary.com/inverse-square+law>.
- [60] Nowostawski, M. & of Computer Science, D. Course - mobile/wearable programming - imt3673. URL: <https://www.ntnu.edu/studies/courses/IMT3673/2017#tab=omEmnet>.
- [61] Bishop, G. & Weimer, D. M. August 1986. Fast phong shading. *SIGGRAPH Comput. Graph.*, 20(4), 103–106. URL: <http://doi.acm.org/10.1145/15886.15897>, doi: 10.1145/15886.15897.
- [62] Stamminger, M. & Drettakis, G. July 2002. Perspective shadow maps. *ACM Trans. Graph.*, 21(3), 557–562. URL: <http://doi.acm.org/10.1145/566654.566616>, doi: 10.1145/566654.566616.
- [63] Sanchez, C. Cpp14.g4. URL: <https://github.com/antlr/grammars-v4/blob/master/cpp/CPP14.g4>.
- [64] Weill, C. cweill/gotests. URL: <https://github.com/cweill/gotests>.
- [65] Wikipedia contributors. 2019. Best coding practices — Wikipedia, the free encyclopedia. [Online; accessed 14-May-2019]. URL: https://en.wikipedia.org/w/index.php?title=Best_coding_practices&oldid=890648825.
- [66] Kosara, R., Healey, C. G., Interrante, V., Laidlaw, D. H., & Ware, C. 2003. User studies: Why, how, and when? *IEEE Computer Graphics and Applications*, 20–25.
- [67] Sridhar, M. 1995. Understanding the user - why, what and how? *Library Science with a slant to Documentation and Information Studies*, 32(4), 151–164.
- [68] Morris, K. 2016. *Infrastructure as code : managing servers in the cloud*. O'Reilly Media, Sebastopol, CA.
- [69] Parr, T. antlr/antlr4. URL: <https://github.com/antlr/antlr4/blob/master/doc/faq/parse-trees.md>.

A Initial Project Description

Title: 3D Representations of Complex Data Structures

Level: BSc

Study program: BPROG

Supervisor: Christopher Frantz

Department: IDI Gjøvik

Project relevant: Excited

Description

Understanding existing software in the context of software engineering is often challenging, especially when employing complex data structures as found in graphics or data-centric applications. Similarly, when subject to the choice of libraries, implementation problems only become apparent once using the library (i.e. after adopting it).

Visual cues can facilitate the faster comprehension of programs, both for novices, but also for advanced programmers, when dealing with complex data representations. This proposal suggests the following:

Allowing the retrieval of existing project repositories, providing an accessible visualisation in 3D, and use this as a tool to understand the visualised program, as well as serving as a diagnostic tool to suggest potential refinements. Optionally or alternatively, the visualisation can be used in VR.

Technical details

- Public-facing Web service that consumes link to repository
- Deployed using container technology (e.g., Docker)
- System retrieves source code from repository
- The system should work with programming languages such as C++ and Java; specific or alternative languages are subject to negotiation
- Where relevant, the system should be able to build the submitted repository
- The system should provide a 3D visualisation of selected data structures (e.g., using OpenGL)
- The developed project should be developed with professionalism in mind (tests, best practices)
- The user of the system should be able to interact with the data (e.g. choosing, collapsing branches of structures, etc.)
- Software quality metrics should be presented to the user
- Alternative direction (depending on student interest): Exploration of data structure in VR, with all other requirements unchanged

Applied project

The product is expected to be used a) to assess the quality of developed software, and b) to explore existing software for quick adoption or evaluation with respect to quality.

B Project Plan

Project plan

Kent Wincent Holt - 473209

Eldar Hauge Torkelsen - 473180

Zohaib Butt - 473219

January 2019

Contents

1	Project goals and constraints	2
1.1	Background	2
1.2	Project Goal	2
1.2.1	Result Goal	2
1.2.2	Effect Goal	3
1.2.3	Learning outcome	3
1.3	Constraints	4
1.3.1	Legal Constraints	4
1.3.2	Technological Constraints	4
1.3.3	Time Constraints	4
2	Scope	4
2.1	Subject area	4
2.2	Boundaries	5
2.3	Task description	5
3	Project organization	6
3.1	Responsibilities and roles	6
3.2	Routine's and group rules	6
4	Planning, followup and reporting	6
4.1	Development process	6
4.1.1	Project characteristics affecting choice of software development process	6
4.1.2	Choice of software development process	7
4.2	Plan for status meetings and decision	7
5	Organization of quality control	7
5.1	Configuration management	7
5.2	Risk analysis	9
5.2.1	Identification and project risk analysis	9
5.2.2	Risk mitigation strategy	9

6 Development plan	11
6.1 GANTT - diagram	11
6.2 Comments on the schedule	12
6.3 Time, Resource and Cost overview	12

List of Figures

1 Tools to be used in the project	8
2 Gantt diagram	11

List of Tables

1 Group roles	6
2 Overview of risks	9
3 Risk analysis form	9
4 Risk mitigation	10
6 Initial story estimates	12
5 Total workhours	12

1 Project goals and constraints

1.1 Background

This project was requested by Frantz, Christopher (product owner) from the Department of Computer Science at NTNU. The project was requested because finding libraries or frameworks that would fit well for new software projects can be a time consuming and difficult process that might require white-box analysis. This project should make the white-box analysis easier by abstracting the code and represent it as a 3D graphical structure and giving information on complexity and quality of the project.

CodePark [1] and CodeCity [2] are two applications trying to solve the same type of problem with similar 3D solution, representing the code as houses or city blocks. CodePark visualize the code base in park-like environment. Each class in the code base is represented as a interactable cube. In the cube, all functions in each class are visible on the wall and upon a click, it highlights the function in the code, which is shown on a separate wall. CodeCity represents the code as navigable cities, where each building represents a class and each district represents a package. The two examples differ from this project in how this project will represent connections related to execution flow and the code not being represented as buildings or cities. This project will also represent different data structures such as vectors and maps.

1.2 Project Goal

We have divided our goal in three sections result goals, effect goals and learning outcomes. Result goals defines what we deliver as the end product. Effect goals describe what the team expect from the system after a set amount of time and are listed to give an indication of the quality the developers should aim for. Learning outcomes describe what the team wants to learning during development and thesis writing.

1.2.1 Result Goal

- Program must, at a minimum, be able to abstract and visualize language version up to Java 11.0.2 and C++17.

- Code visualization must be in 3D.
- Program must give an indication of the quality of the software code.
- Program must be able to take a link to a git repository to use as basis for visualization.
- Program must be able to show potential execution path or program flow through the program.
- Program must have a public facing API for developers.
- Program must be dockerized.
- The user must be able to interact with the 3D visualization to show implementation of data structures.
- The system must present complexity metrics to the user.
- The system must give an indication on where complexity arises.

1.2.2 Effect Goal

1. Quantitative goals for the system:

- The system must handle 30 concurrent users with repository of less than 10 KLOC while using less than 10 minutes of processing time.
- The system must be able to handle a peak of at least 60 users.
- The system must be able to handle repositories of less than 100 KLOC.

2. Qualitative goals for the system:

- The system must help developer(s) gain overview and an understanding of the code base.
- The system must let the user easily navigate through the visualization.
- The system must be naturally intuitive and easy for new users.
- The system must be accessible to people with minor disabilities like colorblindness.

1.2.3 Learning outcome

- Learn in depths WebGL technology.
- Learn professionalism in web development.
- Understand and follow DevOps methodology through out the project.
- Learn in depths dockerizing containers.
- Learn Front-end web frameworks.
- Learn about different code quality metrics for static analysis.
- Learn about professional web architectures.
- Learn about Infrastructure as Code.

1.3 Constraints

1.3.1 Legal Constraints

- The resulting system will not be in violation of general data protection regulation. [3]
- The system will not be in violation of Norwegian copyright act.

1.3.2 Technological Constraints

- The system should be cross platform.
- The system should be dockerized.
- The system should have a REST API.

1.3.3 Time Constraints

- The system should be finalized by 16th May.

2 Scope

2.1 Subject area

Our software is meant to be used in a software development setting. Software development includes many different processes such as system specification, software reuse, integration, development, testing and maintenance. Our system will mostly help with the software reuse and integration part, but will also indirectly affect the other processes.

System specification deals with identifying what the system should do and the environment it acts in.

Software reuse takes those specifications and identifies preexisting systems that could be used instead of developing new software or could be used to make development easier. This process often identifies libraries and frameworks that can be used. Using libraries or frameworks in the systems is often recommended [4] in both large and small systems. Developers spend a lot of time debugging, securing, documenting, structuring, finding suitable libraries, finding frameworks, etc. Finding libraries and frameworks to be used in a project can be frustrating and time consuming if there is no good documentation, source code or life cycle provided [5].

Software integration takes the preexisting systems and either mold them or the software it integrates with to fit smoothly. The integration can vary in difficulty depending on the implementation of each software solution.

Software development creates the functionality that is not provided by preexisting solutions. It is recommended to have as little code complexity [6] as possible for readability [7] which can be hard for developers to see while programming and is partially reliant on how well the software integration went. Potential problems with the preexisting systems are likely to emerge at this state as it is difficult to see limitation and bugs during the software integration phase. Finding bugs in this scenario is time-consuming [8] and might require the developers to do white-box testing on an unfamiliar system. Bugs that are not found during development will hopefully be found during software testing.

Software testing validates that the functionality works as expected with no side-effects. This can help spot bugs or shortcomings in the preexisting systems. If bugs are not found during software testing, then they are likely to persist through the life cycle of the software or until identified by the user

Software maintenance deals with updating the software to remove issues found by users or through normal use. It also assures that dependencies stay up to date.

The problem our system will help with is giving an impression of the build and quality of libraries and frameworks that are considered by the software reuse process as a potential addition for their project. The system will act as a lesser alternative to documentation or as a quick overview to the libraries or framework.

2.2 Boundaries

The system is not meant to help with software specification or the later stages of development after software integration. It is not meant to supersede or be an exhaustive substitution of documentation, but rather an abstraction and quick overview of the code base. The abstraction will be limited to a 3D visualization and textual representation of metric and meta data. It will not change the software in question or improve the quality, but give the end user an insight into the software model and code quality. The project will not involve developing new quality metrics, but use preexisting metrics that have been used by other developers and been proven to be help-full.

The system will not build, nor execute the code base in question but rather do static analysis on the code base. There will neither be provided any test coverage nor execution of any shipped testing functionality.

2.3 Task description

The application should assist in getting an overview over the structure and complexity of the product in question.

The program should take in a Git repository link and/or code base for analysis. The analysis must contain information about code such as lines of code, functions, classes, objects, templates, name-spaces, tree structures, multidimensional maps and relationships between structures. The gathered information should then be abstracted to give relevant and standard quality metrics whilst also showing a 3D representation of the program.

The 3D representation must be understandable and relevant for a software developer. As this is not a well defined metric, targeted user testings must be performed to verify that this criteria is met and must indicate that the system is user friendly and helpful. The metrics to be shown and form of visualization should be determined by the development team through the development process.

The 3D representation should include name of functions and structures for ease of localizing the structure relative to others.

The 3D visualization should be navigable by the end user. The navigation should be intuitive and able to get to important aspects of the program. Navigation should include free camera movement, on track movement and jump/teleport. Track movement should be able to follow connections or relations from function or class, without ending up off target. Jump/teleport movement should allow user to click and get to data structures and code, which can be seen from the users navigational position.

The 3D visualization should also be interactive, letting the user collapse and expand functions, classes and other data structures to show more or less information about them. Information that the user should be able to hide or show including code implementation and metadata. Clicking a data structure should highlight connected data structures. Hover over a data structure should show comments connected to the data structure.

The program should display common quality metrics like coupling and cohesion. The metrics should include ideas from cyclomatic complexity and, where sensible, connascent metrics. Based on the quality metrics, the program should give an estimate of code quality.

Program modularity and scalability is preferred. If possible and time allows, then the program should use docker technology.

The program should be cross platform and requires very little or no installation for end users. It is expected that this will be solved by making it a web application but this is not a requirement.

3 Project organization

3.1 Responsibilities and roles

Group roles	
Member	Role
Kent Wincent Holt - 473209	The Group leader The lead programmer Software developer/tester The release manager
Eldar Hauge Torkelsen - 473180	DevOps evangelist Scrum master The automation architect Software developer/tester Lead security engineer Recorder The Lead Interaction designer
Zohaib Butt - 473219	Software developer/tester The experience assurance (XA) professional Facilitator The utility technology player

Table 1: Group roles

3.2 Routine's and group rules

All group members are expected to work from Monday to Friday from 08.15 to 16.00. If the group sees it necessary to work overtime then the group should work past 16.00 and on Saturdays.

From Monday to Friday, lunch breaks are expected to take less than 45 minuets and the timing is up to the individual and and breaks should be taken when the individual member feels that a break would increase efficiency in the long term or a break is necessary. Sunday is considered a day off, but if a group member wants to work voluntarily, they are allowed, but can't make any claims/excuses related to the work done. Voluntary work should be tracked and labeled as such. Members are encouraged to use Sunday as a break day.

When possible, meetings should be recorded and a meeting report should be created.

If a group member lose 3 days of work without valid reasoning, then a warning will be given. If the warning is unheeded the supervisor will be included and will decide on further actions.

4 Planning, followup and reporting

4.1 Development process

4.1.1 Project characteristics affecting choice of software development process

The system needs to be developed in a short amount of time and the system specifications are partially ambiguous, requiring refinement during development. This is an effect of the research focused topic and would benefit from an agile development method. The project can be considered a success even if it lacks minor features so a development method allowing for iterative improvements would be beneficial. The product owner has given the impression that it would be nice with a prototype version that could be used before the final version.

The development team consists of three persons with only moderate experience so a methodology allowing for low boundaries for communication and close teamwork would work well.

4.1.2 Choice of software development process

Due to the ambiguous specification of the project paired with the fact that changes will take place. Agile methodologies gives us the flexibility needed for the project. Weekly meetings with product owner and user studies will continuously affect the project. As a team we also want to have fixed routines and structure. Therefor the team has decided to use Scrum as a system development process along with DevOps for continuous integration

To have good documentation we will automate documentation using APIDOC, GoDoc and JSDoc tools. For further documentation we also focus on ideas like Infrastructure as Code (IaC) for documenting architecture and environment. Using these tools and IaC along with scrum requirements for regular meetings and reporting and DevOps gives us a good platform to conduct project in agile but also in a structured manner.

The team will setup backlog with user stories which are acquired from user studies. The team has planned to have weekly sprints. Sprint will start on Mondays and end on Fridays. Longer sprint length for a bachelor project does not seem feasible since this will give less opportunities for the product owner to give his input and give us less time to do user studies. There can be some variations in the sprint length if the team sees workload for sprints being off or scrum master sees a benefit in changing it.

For each sprint we will have planning, review and retrospective meeting. Sprint planning meetings will be used to discuss what the team would like to achieve using user studies for the upcoming sprint. There will be held planning poker sessions for estimating user stories to work with in the upcoming sprint. Sprint review meeting will be used to present what each team member has done in sprint and will have status update ready for the meeting with product owner. Sprint retrospective meeting will be used to discuss what went well in the sprint, what could be improved and also discuss estimate precision.

4.2 Plan for status meetings and decision

After each sprint group should have a meeting with Frantz, Christopher (product owner) every Monday at 13.15. In this meeting product owner will give his input on the update which will be used as user stories. We will also discuss the next sprint with product owner. Group should have a meeting with Kolloen, Oivind (supervisor) every Monday at 09:00. Group meetings for code review, discussing internal decisions and planning for meeting with supervisor and product owner is every Friday from 14:00 to 16:00. Sprint planning meeting will be held after the meeting with Frantz, Christopher (product owner).

5 Organization of quality control

The group is expected to have high standards for code quality. All members should follow the official guidelines for [Golang style and code review](#). The w3schools style guides should be used for [HTML5](#) and [JavaScript](#). The CSS-tricks style guides should be used for [SASS](#).

Each member will document and create tests for the functionality they add. Additional documentation is expected for API endpoints such as http GET and POST requests.

The group code review should be done during the Friday meetings and for every pull request. User testing will be done on a regular basis.

5.1 Configuration management

The main focus when setting up the development environment and use of technologies is to allow for seamless connecting between code, workflow and documentation. Considering the agile development method, the tools

must facilitate communication and actors with multiple roles. The agile development also calls for user studies and testing, this makes DevOps particularly advantageous.

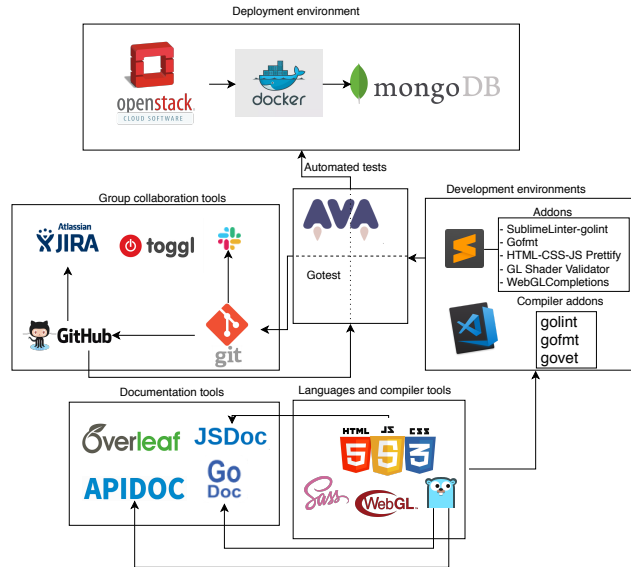


Figure 1: Tools to be used in the project

As shown in figure 1, Openstack with Jenkins will be used for rapid deployment and Infrastructure as Code to seamlessly document environment configuration. Docker will allow for further environment documentation and release on foreign systems. Jira with Github integration will be used to manage tasks. The Jira workflow is set to include issue status as "To Do", "In Progress", "Pending Review", "In Review" and "Done". The stories are estimated by the development team at inception and before a sprint using agile poker. Work is planned through Slack and work hours are logged using Toggl. Editors and IDEs choice is up to the individual but linters and formatters are required to validate a push to git. Languages are chosen to reflect the standards or allow good collaboration and high readability. Go was chosen in particular for its strict syntax and integration with tools such as GoDoc and gofmt as well as a built in testing framework. For seamless connection between code and documentation GoDoc, APIDOC and JSDoc were chosen for their respective language and use case. APIDOC will be used extensively to generate a comprehensive and thorough documentation on the REST API. Insufficient documentation will not be accepted by merge request and feedback on merge requests will ensure that all users follow the same standards and can discuss when different opinions on the matter surfaces. Additional documentation, like wiki pages for structural overview and meeting reports for covering decisions and development processes will be created using \LaTeX and Overleaf for collaboration.

The documentation will hence tightly follow the development process and allow the developer to document their code whilst writing it, the infrastructure will be documented as a side effect of the DevOps automation and the development process will be documented through references to issues.

Automated testing will be done using tools like gotest and AVA to ensure the code on Github is functional and validated before deployment on OpenStack.

5.2 Risk analysis

5.2.1 Identification and project risk analysis

The table 2 contains possible risks based on [1] and compared with the system requirements. The probabilities and effects are estimates done by the core team. The priority is based on the minimal Manhattan distance from top right corner of table 3.

Id	Risk Type	Possible Risk	Priority	Probability	Effect
1	Organization	Losing important data	8	Low	Significant
2	Technology	Technology components aren't scalable	4	Medium	Severe
3	Technology	Insufficient security for the system (Information security)	5	Medium	Significant
4	Technology	Application instability	6	Low	Severe
5	Organization	Team members are unavailable for longer period of time	7	Medium	Significant
6	Requirement	Discovering problems in requirements at delivery	1	High	Severe
7	Estimate	Project size is under estimated	3	High	Significant
8	Organization	Learning curves lead to delays	2	Very High	Moderate
9	Technology	Integration testing environments aren't available	10	Low	Minor
10	Organization	Fail to follow software methodology	9	Medium	Minimal

Table 2: Overview of risks

	Minimal	Minor	Moderate	Significant	Severe
Very High			8		
High				7	6
Medium	10			3, 5	2
Low		9		1	4
Very Low					

Table 3: Risk analysis form

5.2.2 Risk mitigation strategy

Table 4 shows the mitigation strategy for the project risks with highest priority and should give an overview of mitigations that should help with most of the identified risks, as several can be mitigated with similar strategies.

Priority	Risk	Mitigation
1	Discovering problems in requirements at delivery	<ul style="list-style-type: none"> • Use an agile development process • Perform regular user testing
2	Learning curve lead to delay	<ul style="list-style-type: none"> • When a new technology is to be use a workshop for exploring it will be held
3	Project size is under estimate	<ul style="list-style-type: none"> • Use an agile development process • Have regular meetings with product owner
4	Technology components are not scalable	<ul style="list-style-type: none"> • Use IaC and code for modularity
5	Insufficient security for the system (Information security)	<ul style="list-style-type: none"> • Test coverage of all important functions • Use of static analysis tools
6	Application instability	<ul style="list-style-type: none"> • Unit test coverage of all important functions • Use DevOps with live server throughout the development
7	Team members are unavailable for longer period of time	<ul style="list-style-type: none"> • Focus on reviewing git pull requests well • Good communication amongst developers

Table 4: Risk mitigation

6 Development plan

6.1 GANTT - diagram

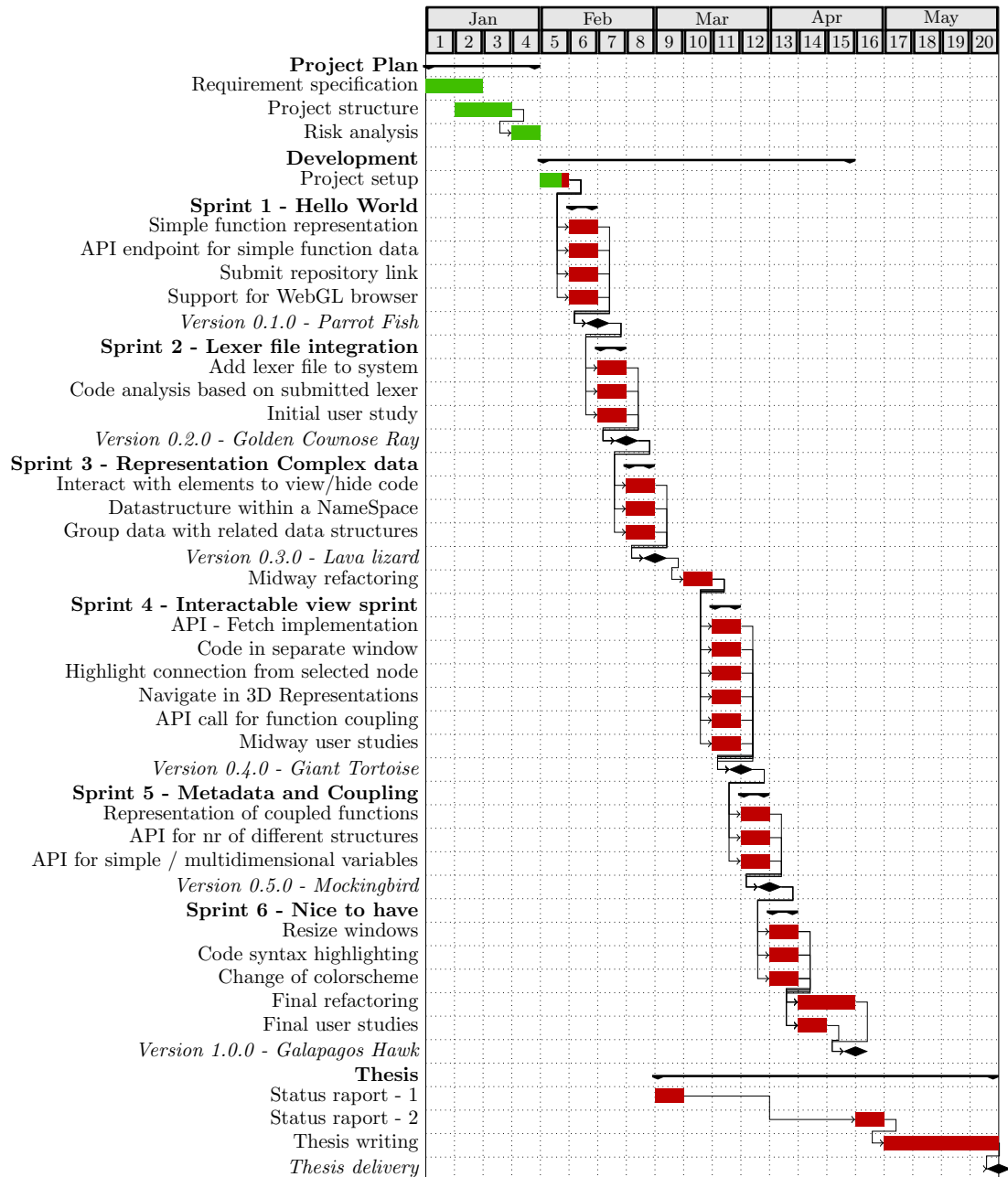


Figure 2: Gantt diagram

The diagram shows the timeline from left to right of the entire project. Green is used to show completed progress while red is used for what is not completed.

Summary	Issue key	Issue Type	Sprint	Story Points
As a developer i would like the API endpoint extended to give enough to represent simple and multidimensional variables.	COD-7	Story	Metadata and coupling sprint	8
As a developer i would like the API to include request for implementation based on function/class/etc name or start/end markers	COD-25	Story	Interactable view sprint	3
As a developer i would like an API endpoint giving metadata about the project.	COD-13	Epic		
As a developer i would like the API endpoint extended to give information about what functions call what.	COD-9	Story	Interactable view sprint	3
As a user I would like to see what functions are tightly coupled in the 3D representation	COD-8	Story	Metadata and coupling sprint	5
As a user i would like to navigate the 3d representation.	COD-10	Story	Interactable view sprint	8
As a developer i would like the API endpoint to include comment coverage and test coverage	COD-15	Epic		
As a user i would like to submit a git repository link for the system.	COD-6	Story	Hello world sprint	3
As a user i would like to view or hide the implementation of a function or class by clicking it.	COD-11	Story	Represent complex data sprint	5
As a developer i want an API endpoint that gives me enough to represent a basic function.	COD-4	Story	Hello world sprint	13
As a user i would like to see a representation of a simple function	COD-3	Story	Hello world sprint	8
As a user I want to submit hello world repository and visualize my code in 3D	COD-2	Epic		
As a developer i would like coding language for analysis to be based on provided lexer file.	COD-24	Story	Lexer file integration sprint	5
As a user i would like to submit a lexer file for defining language syntax	COD-23	Story	Lexer file integration sprint	13
As a user i would like to change the colorscheme.	COD-18	Story	User experience sprint	3
As a user i would like code to be displayed with syntax highlighting	COD-22	Story	User experience sprint	5
As a user i would like to resize windows	COD-20	Story	User experience sprint	8
As a user i would like to highlight connections from selected node.	COD-12	Story	Interactable view sprint	3
As a user i would like to display code in a separate window	COD-21	Story	Interactable view sprint	3
As a user i would like to see test coverage and comment coverage for Golang	COD-26	Story		
As a user i would like the data structures in the 3D visualization to group together with related structures	COD-19	Story	Represent complex data sprint	8
As a user i would like to see what namespace data structures are in.	COD-17	Story	Represent complex data sprint	3
As a user i would like to be notified if my browser does not support WebGL or this application	COD-5	Story	Hello world sprint	2
				112

Table 6: Initial story estimates

6.2 Comments on the schedule

The schedule is flexible and should be changed whenever the priorities change. It might be altered depending on the user studies, product owner changing requirements or design problems. The intent is to use scrum where sprint lengths vary and incomplete stories can be put in the backlog or added to the next sprint. The project features are expected to be defined as user stories refined from user studies results. The user studies take precedence over the developers initial understanding of the users needs and priorities.

6.3 Time, Resource and Cost overview

As of now there are no expected costs. Our resources are limited to workhours and 3 employees only.

Member	Allocated workhours
Kent Wincent Holt - 473209	400 h
Eldar Hauge Torkelsen - 473180	400 h
Zohaib Butt - 473219	400 h
	1200 h

Table 5: Total workhours

The initial story estimates in table 6 resulted in 112 story points. The allocated time of 1200 workhours results in 10 hours per story point. The estimates will be re-estimated on the beginning of relevant sprint and the stories will change.

References

- [1] Khaloo P, Maghoumi M, II EMT, Bettner D, Jr JLL. Code Park: A New 3D Code Visualization Tool. CoRR. 2017;abs/1708.02174. Available from: <http://arxiv.org/abs/1708.02174>.
- [2] Wettel R. Software systems as cities. Università della Svizzera italiana; 2010.
- [3] UNION EORFDE. Lov om behandling av personopplysninger (personopplysningsloven) EUROPAPARLAMENTS- OG RDSFORORDNING (EU) 2016/679 av 27. april 2016 om vern av fy-

siske personer i forbindelse med behandling av personopplysninger og om fri utveksling av slike opplysninger samt om oppheving av direktiv 95/46/EF (generell personvernforordning) [GDPR]. <https://lovdata.no;2016>.

https://lovdata.no/dokument/NL/lov/2018-06-15-38/KAPITTEL_gdpr#KAPITTEL_gdpr.

- [4] Krueger CW. Software Reuse. ACM Comput Surv. 1992 Jun;24(2):131–183. Available from: <http://doi.acm.org/10.1145/130844.130856>.
- [5] Mileva YM, Dallmeier V, Burger M, Zeller A. Mining trends of library usage. In: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM; 2009. p. 57–62.
- [6] Nguyen-Duc A. The impact of software complexity on cost and quality - A comparative analysis between Open source and proprietary software. CoRR. 2017;abs/1712.00675. Available from: <http://arxiv.org/abs/1712.00675>.
- [7] Spinellis D. Reading, writing, and code. Queue. 2003;1(7):84.
- [8] Westland JC. The cost of errors in software development: evidence from industry. Journal of Systems and Software. 2002;62(1):1–9.

C Project Agreement

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

_____ (oppdragsgiver), og

_____ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra _____ til _____ .

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:

- Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
- Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.
6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.
7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.
8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.
9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.
10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.
11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): _____

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): _____ dato _____

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.

Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.

Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

D User study

Norwegian

- **Introduksjon** - "Hei, Jeg representerer en bachelor gruppe some har jobbet med å representere codebaser fra git repositories."
- **Be de lese kode eksemple** - "Vennligst les denne kode snutten i c++. Hvis du har noen spørsmål, gjerne spør. Det er ikke deg vi tester, vi tester systemet."
- **Be de tegne strukturen** - "Vennligst tegn hvordan du ville ha visualisert datastrukturen og forklar tankegangen din."
- **Be de sende inn en Git URL til systemet** - "Dette er hjemme siden for systemet. Kan du prøve å representere følgende Git repo: https://github.com/zohaib194/COVI_test01.git"
- **Få dem til å hente implemntasjon av main() funksjon** - "Er det mulig om du kan finne kilde koden i main funksjonen innen dette systemet?"

English

- **Introduction** - "Hi, I represent a group working on a graduate project, trying to represent code-bases from git repositories."
- **Have them read the code-example** - "Please read this c++ example. If you have any questions don't hesitate to ask, as we are not testing you, but the system."
- **Have them draw the structure** - "Please draw how you would represent the code-base and also explain why you represent the code like you do."
- **Have them submit the git URL** - "This is the home page of the system. Could you try to use it to get a representation of the code at the git repository: https://github.com/zohaib194/COVI_test01.git"
- **Have them get the main() implementation** - "Is it possible for you to view the source code of the main function within the system?"

Questionair

- **How confident are you with the object-oriented paradigm?** - From 1 to 5.
- **How confident are you with C++?** - From 1 to 5.
- **How useful is the information on the homepage?** - From 1 to 5.
- **How intuitive is it to navigate within the visualization?** - From 1 to 5.
- **How easy is it to find the implementation / source code for a function?** - From 1 to 5.
- **How intuitive is the visualization?** - Long text answer.
- **Do the shapes and color differentiate the various data structures clearly?** - From 1 to 5.
- **Do think a customizable GUI would improve the usability?** - From 1 to 5.
- **Suggestions for customizable GUI (if any)** - Long text answer
- **How likely is it that you will have a use for such a system in the future?** - From 1 to 5.
- **Any features you feel are missing?** - Long text answer

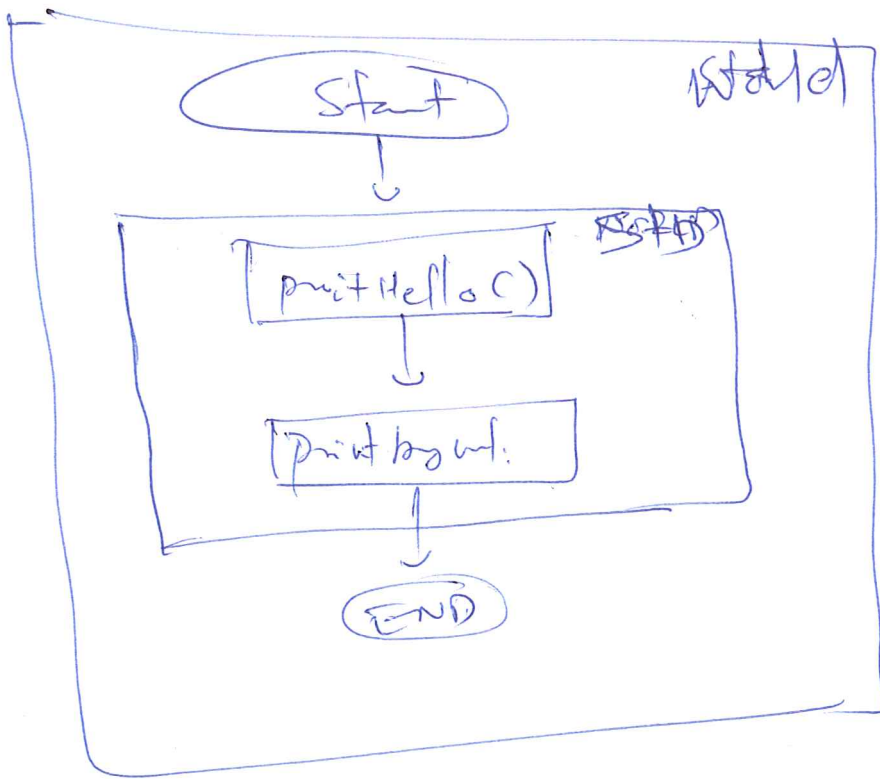
- **Any additional feedback on the entire project.** - Long text answer

D.1 Participation notes 1

Participant: Professional.

- **Have them read the code-example** - Participant did not have any problem reading the code.
- **Have them draw the structure** - The participant started with drawing the execution flow of the main function using a circle and called it "start", then added two boxes and said this was the two functions side the "World" namespace and drew an arrow from main to the first function main called, then from that function to the second function. Then from the second function to a circle named "end". After this it was mentioned to visualize the "World" namespace, the participant drew a box surrounding the two functions within it and called it "World" at first but then shifted this to be the "std" namespace and draw another namespace box around the whole visualization and called it "World".
- **Have them submit the git URL** - URL submission went without problem and seemed to be intuitive although the participant used the cached input on the URL field and did not type it fully.
- **Have them get the main() implementation** - Participant was able to find implementation of main. Participant wasn't able to find the implementation of the other functions.
- **Participant visualization**

1.



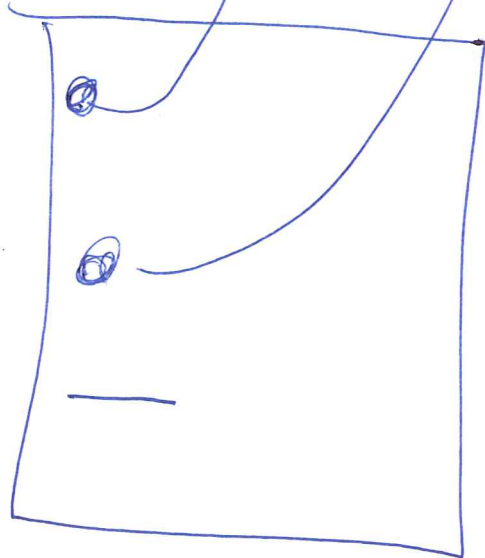
D.2 Participation notes 2

Participant: Student

- **Have them read the code-example** - Participant did not have any problem reading the code.
- **Have them draw the structure** - Participant separated the two called functions into each their circle and added names to each of them. Participant then represented main as a square and drew arrows from main to each function and explained that they were function calls.
- **Have them submit the git URL** - Participant wrote into the git repository field and submitted without problem. Afterwards, the participant used mouse to navigate successfully.
- **Have them get the main() implementation** - Participant was able to find implementation of both main and other functions.
- **Participant visualization**

~~PRINT~~ HELLO WORLD

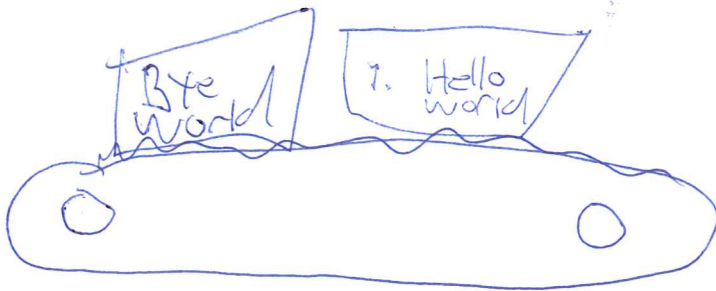
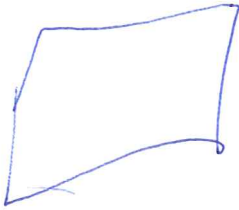
~~PRINT~~ BYE WORLD



D.3 Participation notes 3

Participant: Student

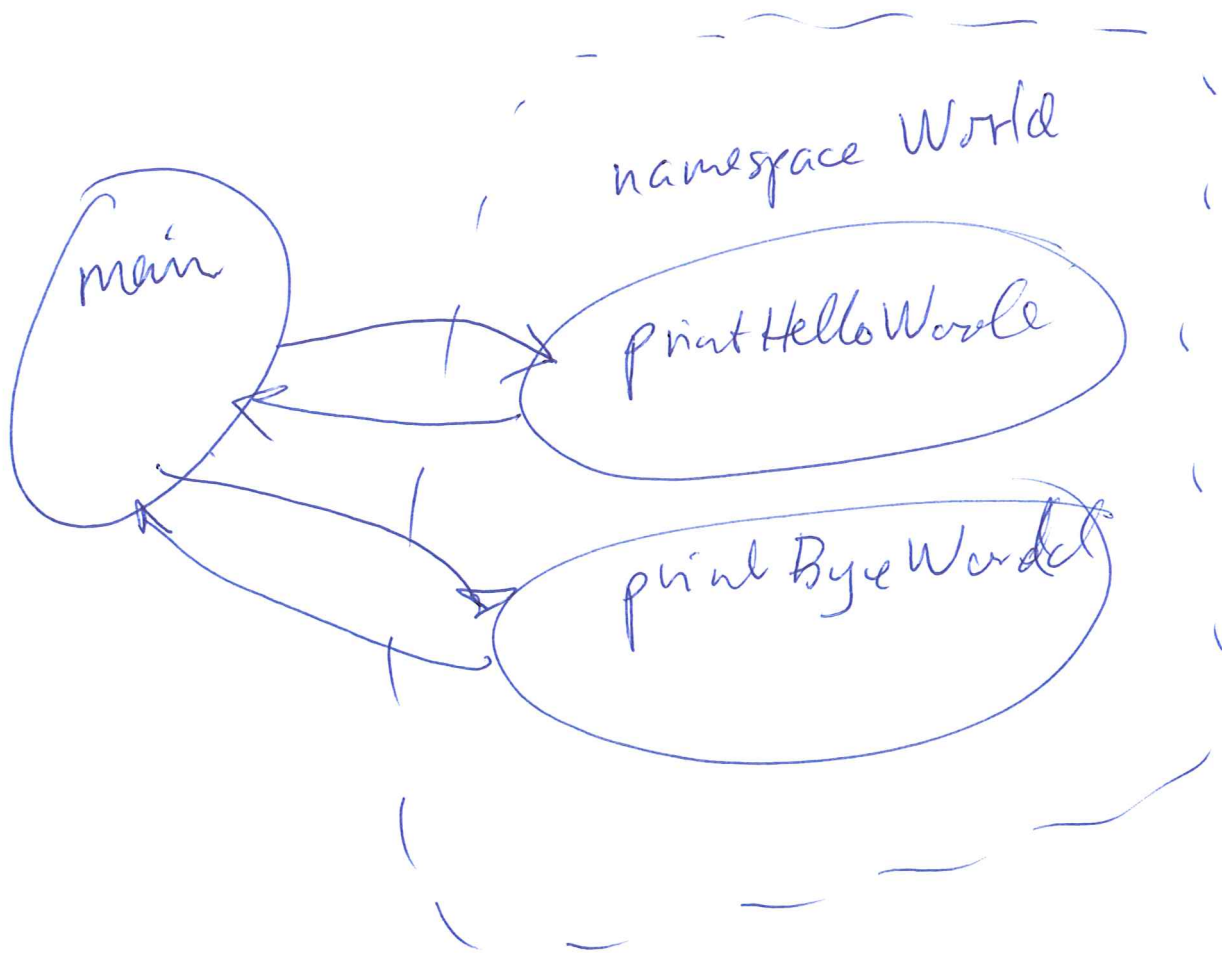
- **Have them read the code-example** - Participant did not have any problem reading the code.
- **Have them draw the structure** - Initially the participant seemed confused by the question, but started drawing to explain the code structure. User represented the code as a conveyor belt with packages on top. the packages were the two functions being called by main.
- **Have them submit the git URL** - User wrote in the URL and submitted. Participant then used the mouse for navigation and note that lines represent function calls and that the representation makes somewhat sense.
- **Have them get the main() implementation** - Participant mentions that the participant does not know at all if task is possible, but continues for a while before clicking the function. Despite the implementation being shows, the participant is unaware and answers that participant does not know.
- **Participant visualization**



D.4 Participation notes 4

Participant: Teacher

- **Have them read the code-example** - Participant mentions that participant has very limited experience with C++ in particular.
- **Have them draw the structure** -
 - - User explains that there is a main procedure and two under-procedures called by main. Namespace is a concept that the user has little experience with, but draws it as a dotted line around the two sub procedures. Participant then mentions the control flow going from main to the two sub procedures.
- **Have them submit the git URL** - User reads the description before clicking the submit button without content repository field. User then mentions that nothing happen when clicking submit and is unable to continue without guidance.
- **Have them get the main() implementation** - Participant tries to use the top bar menu and does not try the navigation within the expected time. When main is visible, the user immediately clicks it to get implementation.
- **Participant visualization**



D.5 Participation notes 5

Participant: Student.

- **Have them read the code-example** - Participant did not have any problem reading the code.
- **Have them draw the structure** - The participant started with drawing a box and called it "namespace". Within this box, two lines were written inside and was said to be the functions inside the "World" namespace. Then the participant wrote "Main" and drew two half circles on either side and said that this represented the main function. Inside of the half circles 2 lines were written as calls to the first, then second function and then a "return 0" line.
- **Have them submit the git URL** - URL submission went without problem and seemed to be intuitive although the participant used the cached input on the URL field and did not type it fully.
- **Have them get the main() implementation** - Participant was able to find implementation of main. Participant wasn't able to find the implementation of the other functions.
- **Participant visualization**

5.

Namespace

```
void printHelloWorld()  
void printByeWorld()
```

Main

```
call World::printHelloWorld();  
call World::printByeWorld();  
return 0;
```

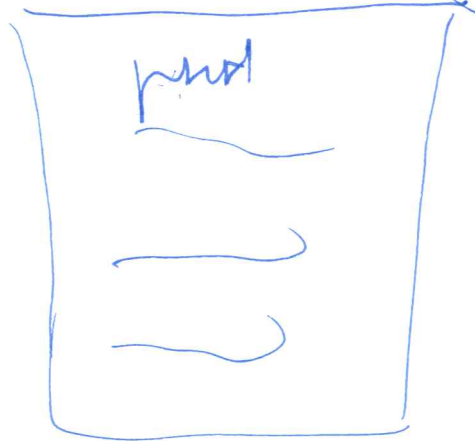
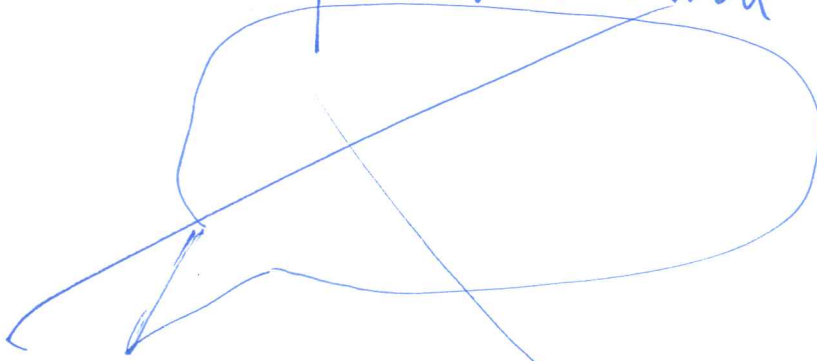
(5)

D.6 Participation notes 6

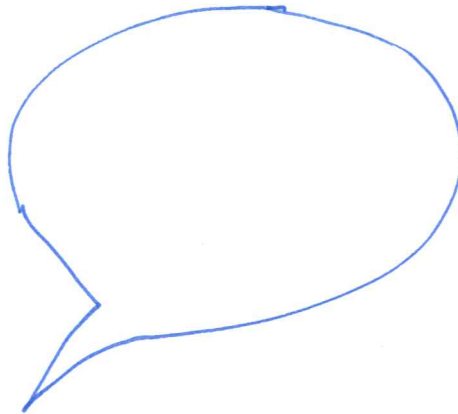
Participant: Student

- *Have them read the code-example - Participant did not have any problem reading the code.
- *Have them draw the structure - The participant focused on the output of the code. Visualized the "cout" as speech bubbles and the code base as a blueprint.
- *Have them submit the git URL - URL submission went without any problem and camera controls seemed to be intuitive.
- *Have them get the main() implementation - Fetching implementation of main function worked without any problem. Participant also used the code on paper and visualization for navigation of calls. Other functions implementation wasn't intuitive to fetch.
- *Participant visualization

print Hello World



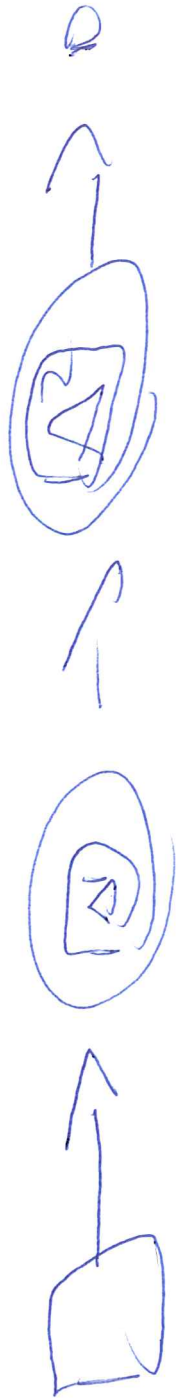
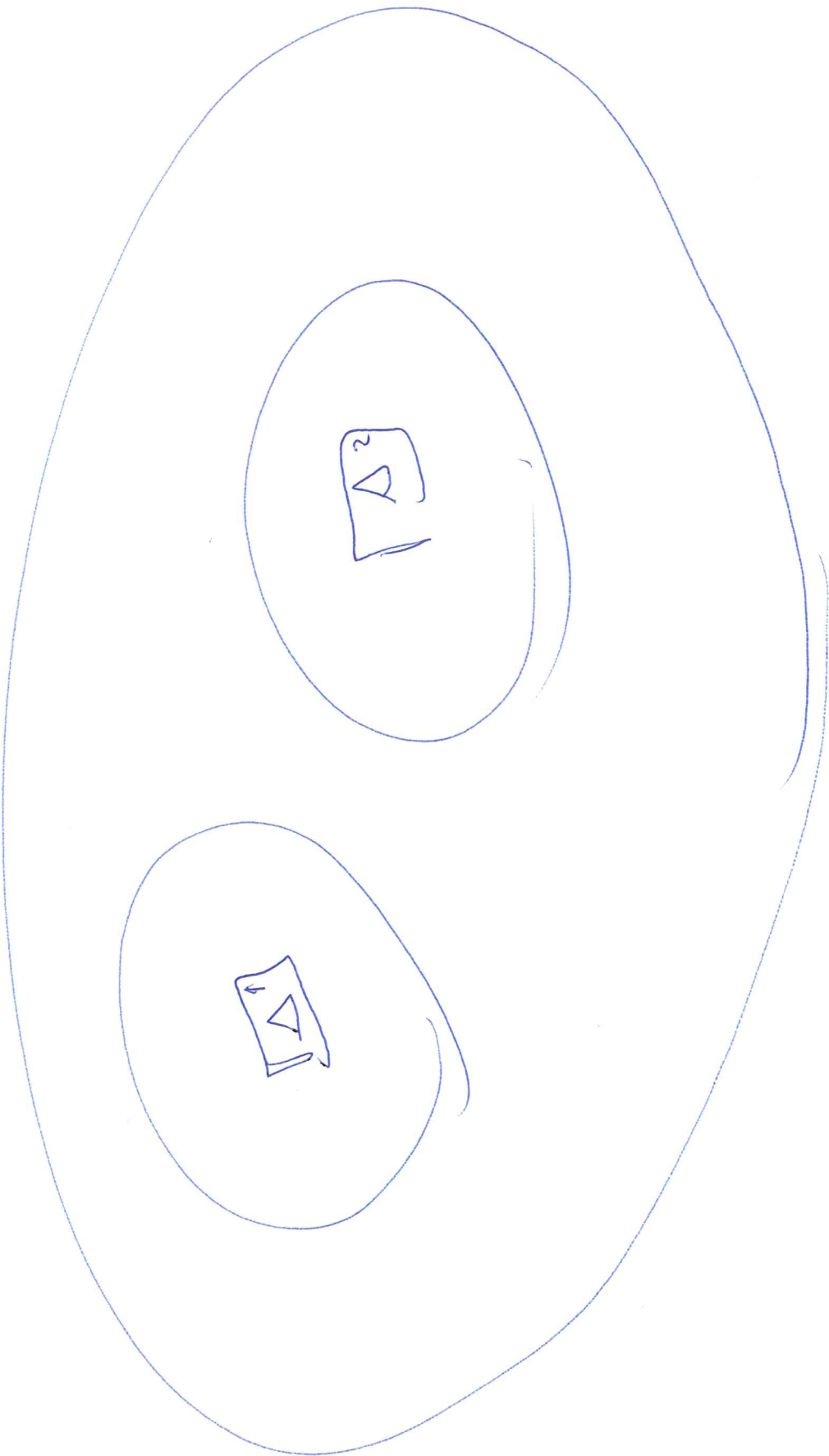
it hau (/



D.7 Participation notes 7

Participant: Professional

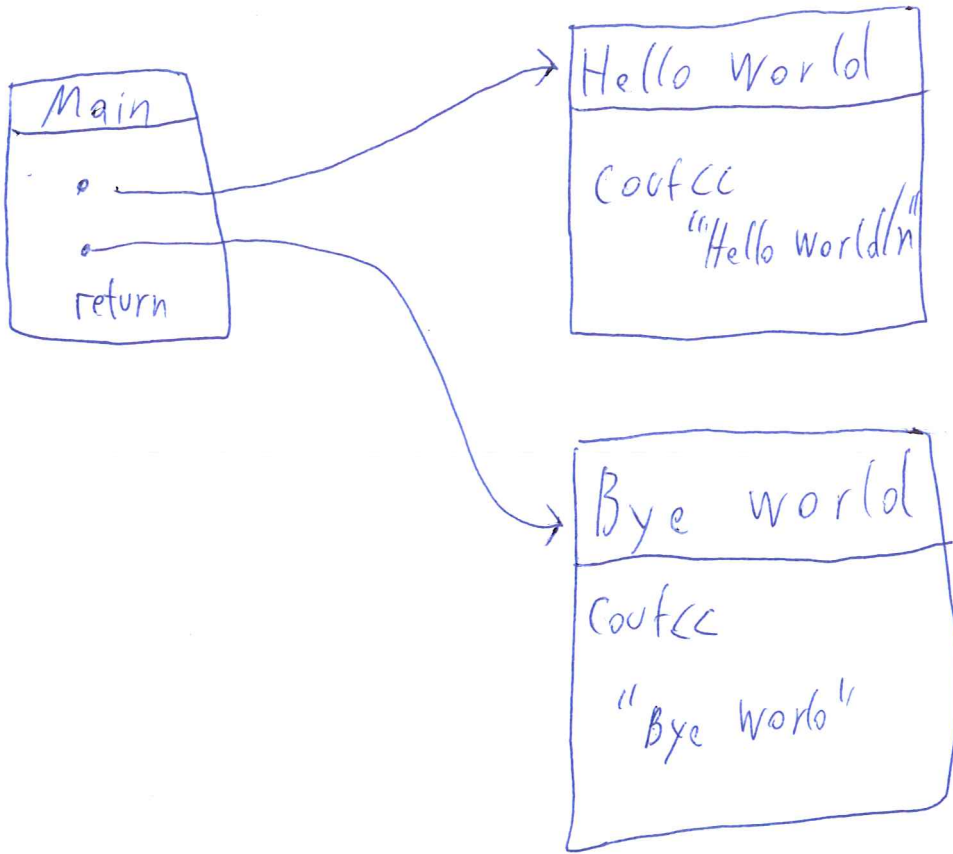
- **Have them read the code-example** - Participant has no issue understanding the code.
- **Have them draw the structure** - User mentions taking inspiration from DFD or Data Flow Diagram when writing. The participant then followed up by drawing the namespace as an oval and the inner functions as circles with square, triangle and a number in each. The user mentioned doing so to identify the different functions. Under the drawing, a pipeline was drawn a square connected to the first function and then to the second function and then on to a circle. The connections were drawn as arrows.
- **Have them submit the git URL** - Participant changed the content of the repository form and used mouse to navigate once the visualization appeared. The user then mentioned that it was a good way to visualize the code.
- **Have them get the main() implementation** - Participant immediately clicked on the function and sees the implementation of main, but struggled to get the implementation of the other functions.
- **Participant visualization**



D.8 Participation notes 8

Participant: Student.

- **Have them read the code-example** - Participant did not have any problem reading the code.
- **Have them draw the structure** - The participant drew three Boxes the title "Main", "Hello world" and "Bye world". Within box "Hello world" and "Bye world" a general form of the implementation of the individual functions was written. In side the main box, the prticipant added two dots one above the other and drew two arrows, one from the top dot to the "Hello world" box and a second from the "Bye world" box and said this was said to represent the function calls in main. After this a line was added underneath the two dots within the box called "Main" which said "return" which was mentioned to be the return of the main function.
- **Have them submit the git URL** - URL submission went without problem and seemed to be intuitive.
- **Have them get the main() implementation** - Participant wasn't able to find implementation of both main and the other functions.
- **Addition notations** - Clicked the list of submitted repositories. The participants was able to scroll in/out. Participants had no problem with the camera controls.
- **Participant visualization**

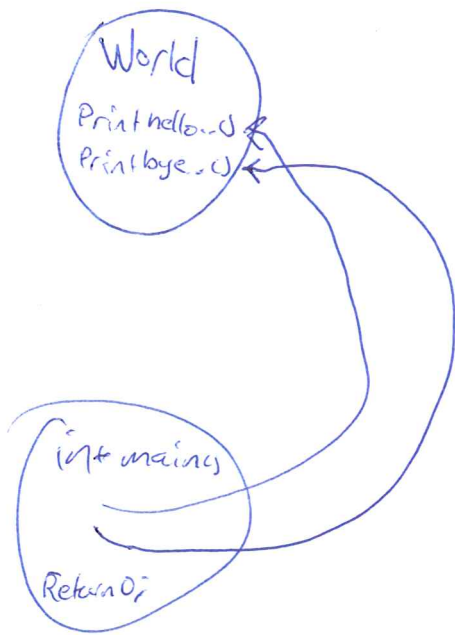


8.

D.9 Participation notes 9

Participant: Student

- **Have them read the code-example** - The user mentions being confused by the task.
- **Have them draw the structure** - The participant explains drawing the namespace as a separate thing with two functions in it. The two functions print and are called by main which is outside the namespace.
- **Have them submit the git URL** - User writes into the git repository field and reads the description before clicking the submit button. The participant then mentions seeing a line before navigating using the mouse.
- **Have them get the main() implementation** - The user clicks the main function and says it is possible to view the implementation, but seem confused when trying to get the other functions. First saying it is not possible before changing opinion.
- **Participant visualization**



E Design

E.1 Initial visualization concept

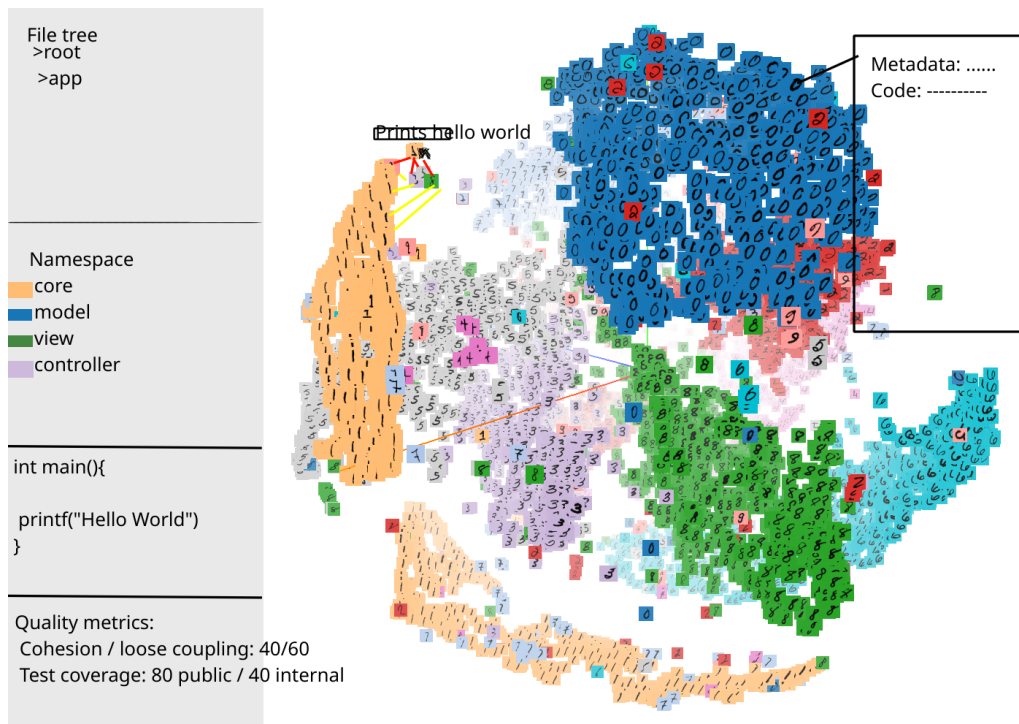


Figure 23: Initial UI concept.

E.2 Logo concepts



Figure 24: Logo for large and small format.

F Jira exported decision log

Decision log

Create decision

Decision	Status	Stakeholders	Outcome	Due date	Owner
Try to implement connascence	DECIDED		Connascence will not implemented		Eldar Hauge Torkelsen Kent Wincent Holt Zohaib Butt
Parsing function calls in c++	DECIDED		Partial parsing without include		Eldar Hauge Torkelsen Kent Wincent Holt Zohaib Butt
Listeners VS Java.String	DECIDED		We use Java.String because the complexity of this issue was too great.		Zohaib Butt Eldar Hauge Torkelsen Kent Wincent Holt
Parse function pointers or not?	DECIDED		The function pointers are not parsed. Currently no listener is found to parse pointers which makes complexity of this task sky rocketing.		Zohaib Butt Eldar Hauge Torkelsen Kent Wincent Holt
Use of websocket	DECIDED		We went with websocket and relaxed the REST requirement		Eldar Hauge Torkelsen Zohaib Butt Kent Wincent Holt
"sloccount" vs. "kloc" vs "wc -l"	DECIDED		We went for "wc -l" which doesn't care for any language.		Kent Wincent Holt Eldar Hauge Torkelsen Zohaib Butt
Antlr prototype	DECIDED		We will be prototyping it and expecting to refactor later.		Eldar Hauge Torkelsen
Using antlr	DECIDED		We went with antlr as it is a well vetted and well regarded project with long history and regular updates.		Eldar Hauge Torkelsen Zohaib Butt Kent Wincent Holt

postponing/removing userstory allowing user to submit lexer files	DECIDED	We prioritised other stories and expect the story to be removed later.	Eldar Hauge Torkelsen Zohaib Butt Kent Wincent Holt
Use of Force directed graph library	DECIDED	We decided to use a self implemented fdg algorithm (based on Kamada & Kawai (1989)) because of library constraints. Cause we represent different data structures (including function, classes and namespaces) as different nodes, some nodes would naturally have a stronger attraction to related structures and more repulsive to non-related. All libraries we found, took one attractive and repulsive force and applied it to all which didn't serve our needs.	Kent Wincent Holt Elda r Hauge Torkelsen Zohaib Butt
Antlr for Java target or Golang target	DECIDED	We decided to use Java to parse and give the output to Golang to further construct the REST API.	Zohaib Butt Elda r Hauge Torkelsen
Platform for the product	DECIDED	We decided to go for web solution because then the solution would be operating system independent.	Zohaib Butt Elda r Hauge Torkelsen Kent Wincent Holt
Backend server language	DECIDED	We went with golang.	Eldar Hauge Torkelsen Zohaib Butt Kent Wincent Holt
Refactor the java parser	DECIDED	Limited refactoring, making models for defining structure and print as json structure. We left the refactoring required for understanding scopes for a later sprint as this require more work.	Eldar Hauge Torkelsen Zohaib Butt Kent Wincent Holt

G Meeting Logs

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others

Absent: Holt, Kent Wincent

Agenda

- Feedback on report draft
- What should be the title of the development model and related topics?
- What should we do regarding the open task description?
- What is the difference between subject area and background section?
- What should we do with the effect and result goals?
- Can we change the project plan during the development?
- What should be added to the appendix?
- What development method should we use with DevOps?
- do we lack any group roles?

Minutes

Feedback on report draft

Not much to say, it is overall ok. Regarding the routines you should add what will happen if rules are broken. An example could be 3 days without working gives a warning before talking with me (supervisor).

What should be the title of the development model and related topics?

You don't need to use the original title, make a new one for yourself.

What should we do regarding the open task description?

Discuss with the product owner, try to extract more information. Talk about what you expect should be the requirements and go from there.

What is the difference between subject area and background section?

There is not much of a difference, what you got now seem to be ok.

What should we do with the effect and result goals?

result goals is what you deliver on the last day whilst effect goals is the effect of the program in action after delivery. You won't have the chance to verify that it's effect are good enough. Don't set error message as a result goals as those are expected anyway.

Can we change the project plan during the development?

Yes, try to make it as specific as possible now, but as you are using an agile development method, you should change it later. Might want to have a old and new version of the plan to compare at the end.

What should be added to the apendix?

You should add meeting summary and time log but not things like commit log or change log.

What development method should we use with DevOps?

That is up to you. Scrum, as you have mentioned, is a good idea but you should look at other alternatives to.

do we lack any group roles?

You only lack those that are specific for the development method you chose.

Internal meeting

Participants

Convener: Holt, Kent Wincent

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others

Agenda

- Preparation for Christopher meeting
- Preparation for Kolloen meeting

Minutes

Preparation for Christopher meeting

We need more concrete specifications for the task in order to write a good report. The problems mostly regard requirements and risks. We should make a drawing of how we see the system and confirm with him that our vision does not contradict his. Might want to mention how the nodes in the 3d view has gravity towards each other and towards meddle based on number of connections and connections to each other.

Preparation for Kolloen meeting

I will be a short meeting as we talked with him on thursday. We need to ask about the gant diagram as we are using scrum. The risk analysis should also be discussed.

We still have to do task devision, plan for handling important risk and development plan. We should probably have done a bit of that before the meeting.

Other

We should probably work more efficiently, we spend too much time talking.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback on report draft

Minutes

Feedback on report draft

Ikke lenge siden sist så det er ikke noe å si. I forhold til rapporten, si ifra hvilke seksjoner jeg skal se på.

Om gant

Det er greit å bruke burndown chart istedet for gant.

Risk managment

Vi har nevnt 10 risk managment og lurte på om du kunne i noe tilbakemelding. Bør nok legge til hva som kommer til å skje hvis noen blir syke.

Hva en ofte bommer på

Ofte bruker folk for lang tid på å lære noe nytt. begynn med det en ikke kan fra før. En har bedre tid til å lære det.

Framework

view.js som frontend rammeverk. Angular er vanskeligere å sette se inn i. Polymer bør sjekkes ut også.

Hva bør gjøres mer på

Fyll ut helt på alt. Før ta dette mer på mandag når siste gjennomgang skal være.

Design i planleggings fasen

Ikke ta med design nå. (ikke legg til bilder nå)

Other

Skal userstudy legges til som vedlegg. Viktigste punkter fra studies kan tas med i rapporten.

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Specific constraints to include.
- Open stack for deployment.
- MIT License.
- Interactions.
- Expected systems effect.
- Worries.
- other.

Minutes

Specific constraints to include

Should check out juristical constraints yourself. should work with gitrepo but would also be nice for it to work on zip files.

Open stack for deployment

Would not like OpenStack as a dependencies, docker would be nice. would be nice to be able to run it on "My machine".

MIT License

Perfect.

Interactions

Open collapse like makes sense in 2D is most important other ideas are also good. Must always be able backtrack.

For the gathering of related datastructures we should check out force directed algorithms.

Expected systems effect

Should be able to be used in a classroom setting where all students use the system at a time.

Worries

You should not focus on worry, just try to learn as much as possible.

Other

For the quality metrics we should check out cyclomatic complexity and connascence metrics.

We should also check out profiling and software quality tools as used by other IDEs like visual studio.

Kotlin would be nice for mobile programming.

Would be nice to have an exposed POSIX API for accessing data without visualizations and be able to choose what information these requests should contain, in the sense of static analysis data, quality metrics data and/or connectivity data.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback on report draft

Minutes

Feedback on the report

The introduction should be more general, making it more specific later on. Changing it from "Finding libraries or frameworks" to "analyzing source code". There is too much use of the word "Should". The report is meant to be binding and contain what must be done. If additional should be added, then that should be clearly separated.

There is a grammatical error in 2.3.1.

In 3.1, when talking about the development process, it gets a bit too loose. You should add where in this section your project acts. Don't just add this in "bounds".

When you mention that the system can be useful for documenting, this conflicts with the usecase being for identifying good libraries and framework.

It's a bit confusing when you use the words "Data structure" and "function" interchangeably.

In terms of risk analysis, you have set that one or more members of group are unavailable for a long period of time as priority 7, but have not set up a mitigation for this. Add priority column on "mitigation table".

Version Lava lizard and version 1.0, before "all development should be done in the middle of April"; It's ok to have time to write on the report. But the quantity you have added is more than usual. Usually one only allocates 2-4 weeks.

Your user studies have been set up with a lot of other stuff. It would be better if it was mentioned how much time is used on that part, as now it looks like there is not enough time.

This is a complex thesis, It is important to do user study in correct manners. User studies is a hard process and we as developers do not have much experience in this topic. Usually the user studies requires 5 - 20 users and more than a week to be properly planned and conduct.

It seems strange that we will be doing all analysis ourselves. Could look into "Sonar cube" for a part of analysis.

It would be better if its added that you will find software to do part of the features for us.

Supervisor will find out the delivery data of bachelor thesis.

Supervisor does not know any automatic usability and accessibility tool.

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

Its ok to say "you don't have webgl, good luck". I would suggest you try to find out what the tricky parts are and chose your libraries around that.

I think it should be made future proof, so don't use angular. Could look into react native but the focus should be on the webgl part. Do the webgl part well and the rest is not important. You need systematic decisioning to show why you went for a framework/libraries.

For the first sprint use it for trying out different things and figuring out what works.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback on report draft

Minutes

Rapporten ble godkjent og virker bra. Viktigste er at dette er et verktøy for dere.

Fristen for innlevering er nå riktig i studentweb, dere vet når dere skal levere.

Det som er vikti er at dere noterer alle avgjørelser underveis. Det er viktig for slutt rapporten. Ja, det kan være nok med sprint review meetings for dette, i og med at dere har ukentlige sprint. Dette er den største feilen folk gjør.

De delene av prosjektet synes er vanskelig, begynn med det. Det som er trivielt kan dere ta senere.

Vi tenker å bruke Threejs er dette bra? svar: Vet ikke, gjør ikke så mye med den delen av utvikling. Bare husk å skrive hvorfor dere tok dette valget. hvis det viser seg å ha svakheter er det da enklere å forklare til slutt.

Bruk produktet dere lager til å sjekke alternativer for Three.js for demonstrasjon.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others:

Absent: Holt, Kent Wincent

Agenda

- Feedback on report draft

Minutes

Dere ligger bak, men det er ikke unormalt etter første sprint. Det er ikke så stort problem hvis dere kan hente dere inn og når dere bruker 1 ukes sprint.

Det at devops delen ikke er oppe er ikke noe stort problem, det kommer.

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

As an opening, we estimate that we are 1 day behind. response: Thats not a worry, there will be more like that.

We had some problems with imgui, The functionality is there but the boxes are black. We should talk to Per Morten as he has experience with imgui, but not with js binding.

We had problems with antlr, it was made in java but has golang target but that is in development. We went with java version.

We have set up the server, no testing yet and it took longer than expected.

We will have problems that others don't as your project is entierly different from the other projects. Don't worry too much.

next sprint will be "representation of complex datastructures". Seems good, don't get too discouraged if you can't make too complex structures. Would be nice with multilevel structures and trees.

Not a fan of npm. Breaks often. Its a possibility but would prefer a lightweight option for important dependencies.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others

Absent: Holt, Kent Wincent

Agenda

- Feedback on report draft

Minutes

Feedback on report draft

Ligger fremdeles back, har hentet in en god del av det vi tapte på første sprint. Ikke brenn all kruttet nå. 70 timers uker greier dere ikke hele veien.

Under sprinten fikk vi problemer med csp/ origin policy.

Hvis vi styrer både server og client, kan vi sette i headeren en ip range. Så det skal være overkommelig. Vi har brukt.

Vi bruker nå en frontend server og en rest api. Vi må fike opp i bruken av http og https.

Vi har ikke fått skrevet noe på rapporten men det skal gå bra hvis vi bare skiver ned viktige avgjørelser. Status rapportene sendes bare til

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Vincent

Absent:

Agenda

- Feedback

Minutes

Update from last sprint, we were 3 days behind. now we are 1 day behind after 212 hours. We can now display something, but currently broken by backend changes.

Currently the 3D force directed graph works, but we don't use backend data yet (soon). The algorithm is not a library, we implemented it but we might change and make it a dependency due to some limitations.

The force is calculated as individual links between each node, other implementations use a global force instead.

There could be complications with this as larger systems might collapse. Will probably need to be revisited often throughout the project. Interesting. "Can of worms". Should bear in mind that attraction change over distance. We have done this.

The json model sent from the backed duplicates with multiple scopes. While looking into it check how to see what you can get from parts, try to do it so you don't need to revisit the schema. We are looking into listeners and visitors.

The parsing takes a long time, should do some caching, and delete things that are not used for a long time.

Including imgui creates a dependency on emscripten and use of npm for now.

We should probably spend time in the upcoming sprint on working on the front-end, refactoring it as now if multiple were to work on it, there is bound to be major conflicts. The Antler scope problem is kind of a major blocker, but with data mocking we should be able to get past that.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- The group worked well and was able to make of for much of the time lost in the last sprint.
- The group was able to integrate GUI framework ahead of schedule.
- Backend was refactored to a stage where it could be used for the sprint.
- The team learnt about antlr quicker than anticipated.

What should we have done better?

- The group was forced to work over-time to the point of exhaustion.
- Could not immediately find a good force directed graph library/algo-rithm that would take the input data we required.
- There was very little time to document code or creating tests.
- The group realized late that their parser implementation was not sufficient.
- The estimations of time reacquired per task was significantly off.
- The tasks conflicted forcing members to do pair programming on relatively simple programming components.

Is there something new that the team wants to start doing?

The team will divide the stories further so that each member can be expected to finish several by the end of the sprint and prevent working in the same part of the code base. The tasks should also be clarified better and be more concise.

Is there something that puzzles us?

We need to look into Antlr visitor versus Antlr listeners. We should look into professional frontend and file structure as well as programming patterns. We might have problems with latency and computational complexity.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback on report draft

Minutes

Feedback on report draft

Nåværende status er at vi fremdeles er 1 dag etter. Dette er ikke noen krise, men tenk på hvor mye dere legger i sprinten. Er nok mer interessant med arbeidsvekten.

Ligger nok litt bak på rapportskriving

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Vincent

Absent:

Agenda

- Feedback

Minutes

We have less to talk about this time as its mostly re factoring. We did not change much in the general structure, we looked up how to structure with a module pattern and android studio like approach with string localization and style. We found a solution for figuring out where in the scope we are using a simple stack and a tree structure. We should document our decisions.

We currently have scopes working for only namespaces and there is duplication on cpp files.

Could have a force directed graph on the captions of the nodes (The name labels).

The size of the objects could be based on the number of lines and complexity.

We should check for informal user-studies or ethic's board for formal user-studies. We need concrete feedback. It seem to be more on the side of informal studies.

We don't need compliance form if we don't collect user-data or want to cite a person directly.

For the next sprint we had planned meta-data and multidimensional variables, we won't be able to finish this but we will go in this direction. We should focus on the meta-data and cohesion, make it easier for users to understand what is happening.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- The stories were divided into multiple small tasks, which made development very efficient.
- The group improved with holding 8 hours limit for work day.
- Code quality became better, as we learned JavaScript and did not care about backwards compatibility beyond WebAssembly.

What should we have done better?

- We should have worked more efficiently with the front-end refactoring.
- The slow refactoring made it hard to complete other tasks.
- The refactoring covered a lot of files and made for a difficult merge.

Is there something new that the team wants to start doing?

We will focus heavier on setting up sub-tasks as smaller tasks proved use-full in previous sprint.

Is there something that puzzles us?

We should look more into user-studies, as we had not considered the ethics board required for formal user-studies.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others

Absent: Holt, Kent Wincent

Agenda

- Feedback on report draft
- What should be the title of the development model and related topics?
- What should we do regarding the open task description?
- What is the difference between subject area and background section?
- What should we do with the effect and result goals?
- Can we change the project plan during the development?
- What should be added to the appendix?
- What development method should we use with DevOps?
- do we lack any group roles?

Minutes

Hva skal vi gjøre for status rapporten? Veldig uformelt dokument. Bare skriv om hvordan dere ligger an. skir om diskusjoner dere har hatt, problemer og aha opplevelser. Vi har fulgt rådet med websocket, det virker som det går greit annet enn at coden virker mye mer uoversiktlig, vi vet ikke helt hvordan vi kan fikse det. Det er et krav om at vi bruker REST, skal snakke med arbeidsgiver for å sjekke om dette er greit. Vi forbedrer oss med scrum via å dele opp oppgaver bedre, men estimering er fremdeles ikke bra.

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

We have used a websocket which is in conflict with the requirement for REST api. REST api criteria has been loosened, the idea is to have it available at all times.

We found that figuring out what functions call what won't be trivial. It can contain nested structures recursively. We will be able to solve it, but it might take a long time to add this feature. We work on separate tasks, but have a low threshold for getting help from other groupmembers.

We should look into interpreter pattern.

If we use external parties for quality metrics, then we need to make sure we can submit code snippets(function level) and not whole projects.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- The team worked in average 8 hours a day.
- Team was able to finish most of the issues for the sprint.
- Team is getting better in dividing big stories into sub small stories.

What should we have done better?

- We should have estimated sub stories to manage time better.
- We should have completed informal user studies.

Is there something new that the team wants to start doing?

We will start estimating sub-tasks as well as stories and epics.

Is there something that puzzles us?

We might have problems relating function declaration to function call and definition when they are in different parts of the program.

1st Statusreport

Participants

Convener: Holt, Kent Wincent

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others

Summary

As of the 8th of March, we are behind schedule due to the complexity of parsing and quantity of new technologies the team had to learn.

We haven't been able to write tests for any functionality, due to functionality taking precedence over tests and the misestimation of stories. Stories have in most cases been underestimated and not divided or concrete enough.

Despite this, the progression is still good and we expect to have a good product by the end of the development phase.

So far we parse and handle namespaces, functions, packages/using namespaces and partially classes for both Java version 9 and C++ version 14.

Challenges

Parsing programming languages turned out to be more complicated than expected. We were using Antlr, but could not use its generated go code due to embedded Java being required in the grammar file to describe the programming language. As an example of Antlr complexity, if you want to find the function identifier/name through a class declaration, then you might have to go through a structure with 10 distinct contexts where 4 are called recursively. Simplifying this was done through handling scopes separately. It was difficult to make the parser in such a way that we could separate scopes and refactored several times ending with us using a stack based implementation. Where stack will give us what scope model we are parsing so we can add the data found in the scope model.

We also underestimated the effort required to do user-studies, so this has been postponed for now. Every time we intended to follow through with our user-studies schedule, we found that slight changes would make the studies significantly informational and help-full. One such slight change was to add links for calls in the 3D representation, but this proved significantly more difficult than expected as to identify overloaded functions and class member functions, we were required to also parse name-spaces and variables with their types.

The adding and parsing took a lot of time for multi-file repositories. The user would see the loading circle without any feedback on status of the system. To fix this we decided to send the API request using websocket protocol so that client and server could communicate while server is parsing.

Accomplishment

We have a working minimum viable product that partially parses functions, namespaces, classes and variables of the submitted repository and displays basic information about the project.

The API is currently partially implemented. It has 4 possible endpoints which is Adding of repository, initial request, Listing of stored repositories and fetching implementation of a function.

For visualization of different data structures we decided to use forced directed graph to show the strong/weak relations between nodes. We are also thinking to apply level of detail, where one can zoom in to a node on the graph and a separate 3D space will be initialized showing a graph containing data structures within the node scope. This will require rework on current implementation of forced directed graph.

Upcoming

We are currently looking into more complicated quality metrics, like Connascence metrics, Halstead complexity measures, Cyclic complexity and programs like Sonarqube that might help us calculate these metrics easier. Later we will change the 3D representation to better reflect scopes in the parsed code and window management to make it easier for end-user to navigate and control the application.

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

We have added a front page with description and supported browsers and current features handled. We have websocket finished and giving more info about status user.

Would be nice to make it possible to compare different repositories. Would be nice to be able to change between different repos and retain state. We need to consider memory at this point.

We will continue with function calls and class parsing. In addition we would like to represent the scopes on front end. In addition we have looked at connacence and cliometric, metrics. Found a related one for Halstead complexity measures.

If we do user studies, we need to try to get around gdpr and have it offline. It would be difficult to go for the entirely formal approach requiring questions to be submitted and confirmed through a process taking over a month.

We are worried about the complexity of figuring out functions as in c++ its tightly connected to the data.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- Stories were described better.

What should we have done better?

- Should have focused more on the stories at hand
- Estimates were far off and the complexity turned out to be significantly higher than expected
- Several stories were set as sub-tasks that should have been stories

Is there something new that the team wants to start doing?

Better division between epics, stories and sub-tasks.

Is there something that puzzles us?

Complexity and flexibility of languages is scary and puzzles us a lot. Amount nested if/else if statements increase day by day and this might cause a problem for unit testing.

Meeting with Product owner

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

We are still struggling with parsing functions, we have refactored from multiple conditionals to instead use listeners. We also try to debug by using reflection to find out what we have available and what it will return.

At this point we can think about standardizing a middle layer to make it easier to add new languages later.

We are thinking to only parse one language from now as we have found that our approach would work for multiple languages without problem. If we continue with only one language from now we could focus on other aspects.

The java grammar is very good and made by the one who made antlr and follow the specifications.

We will still make as much as possible language independent.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- We managed to refactor the back-end to use listeners instead of huge nested if statements code blocks.
- We managed to change the structure on front-end to use trees instead of arrays for force directed graph and parsing.

What should we have done better?

- At no point should the group be expected to work for 200 hours in one week.
- Estimation was far off, the stories that were expected to be almost done from previous sprint are still a problem.

Is there something new that the team wants to start doing?

The team wants to have a sprint that focuses heavily on testing as that has not been prioritized.

Is there something that puzzles us?

Parsing function has puzzled us for the past 3 weeks, as C++ target functions are very flexible and can have anything in the body. We need a way to differentiate the function calls from the pointers assignment. Also keyword such as "this" is considered a function call no matter if the keyword is used to set a member variable or call a member function. Function calls are difficult because they can be overloaded (so the parameter types are important), scoped (so what is in front of the identifier is important), and can be called with objects whose scope is declared elsewhere.

Meeting with Product owner And supervisor

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

Life is good, yet bachelor is going slowly. We have used the week for testing. It goes slower than expected partially because we are using multiple systems with multiple testing environments, (AVA .js, go test .go and j-unit .java). We have some problems with j-unit as its difficult to mock with antlr. We have found a few bugs as a result of testing, like URI validation and unnecessary Fatal instead of error call. We have productively 2 weeks of development left which can cause a problem for the amount of work left. We want to Finnish scoping on front end, function calls, class parsing and cyclomatic complexity. Would be nice with caching as well and possibility of only saving last "x" repose.

We should structure the report to show the testing as a quality assurance and note that everything before the point of test-writing is likely to work.

Both supervisor should probably be on the list, but it should be noted when Christopher became supervisor.

About unfinished quality metrics (connacense), we could mention the possible with definition in the background and mention them later and in the future work having a red thread through it.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- The team managed to finish most of testing related issues.
- The tasks were divided well enough that unexpected hinders allow members to just change over to a different task until problems were resolved.
- The team realized that the complexity of previously written code was higher than expected.

What should we have done better?

- The team took longer than expected to setup testing environment
- Large portions of the previously written code did not lend itself to creating simple tests.

Is there something new that the team wants to start doing?

The next sprint will focus on more complicated tasks so the team will hold pair programming sessions for increased productivity. Team has planned to spend about 8 hours per day on pair programming session til the issues are resolved and spend the rest of the time working on the testing related tasks individually if possible.

Is there something that puzzles us?

We are still worried about estimating the three most complicated tasks. We have tried to estimated them before and been significantly off. We hope the pair programming will help.

Meeting with Product owner And supervisor

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

For the last week we have been pair programming and gotten a lot of difficult tasks done. We should fix the problem where too many nameplates causing clutter. Could substitute clutters with number of nameplates or number of types.

For setting up the thesis, we should structure it systematically like MDA (mechanics dynamics aesthetics), low level implementation or high level overview. We should also connect what we have done with the other subjects we have had.

We should think about delta-parsing but implement the "bigger fish to fry".

We should talk about use of libraries vs standards.

we should talk about complexity metrics and how this correlates with byte-code and not always source code/ face value.

For testing we can take names if they allow us to and quote them. We should allow for change in the rest api and give documentation by version.

We should talk about what we consider feature complete.

Retrospective meeting

Participants

Convener: Torkelsen, Eldar Hauge

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Vincent

Absent:

Agenda

- What went well?
- What could have been done better?
- Is there something new that the team wants to start doing?
- Is there something that puzzles us?

Retrospective

What did we do well?

- The pair programming worked very well for parsing function calls task.
- The team managed to complete two complicated problems that had lasted for several sprints.
- The pair programming helped getting all members up to date with the different component.

What should we have done better?

- The pair programming went slowly when some group members were tired.
- The team should have followed the time stated in project plan a lot better.

Is there something new that the team wants to start doing?

The team will be more open to pair programming and will let tired members work on their own tasks.

Is there something that puzzles us?

We are still uncertain of how long the visualization of function calls will take.

Meeting with Product owner And supervisor

Participants

Convener: Frantz, Christopher

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others: Holt, Kent Wincent

Absent:

Agenda

- Feedback

Minutes

We have gotten our Supervisor back. We are starting on the thesis writing and won't do much with the development. We have gotten some pointers on structure for writing.

For the state of the program, we are having problems with merging files, at least in c++ as it is a compiler level statement and not included in the antlr.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Butt, Zohaib

Recorder: Torkelsen, Eldar Hauge

Others Holt, Kent Wincent

Absent:

Agenda

- Hvordan referere til programm beskrivelse?
- Hvordan referere til ntnu-template?
- Hvordan kan vi oppdatere overleaf cache?
- Hvor skal vi skrive om bruken av module pattern?
- Hvordan minimere kodesnutter?
- Max lengde av kodesnutt?
- Oppset av neste møte?

Minutes

Hvordan referere til program beskrivelse?

De er ikke arkivert på web, så det kan være vanskelig. Gjør så godt dere kan, bruk beskrivelse på nett. Author er emneansvarlig etterføgt av institutt.

Hvordan referere til ntnu-template?

Normal gruppe referering mot github. Authors er man contributors, ikke ha med publisher.

Hvordan kan vi oppdatere overleaf cache?

Vet ikke, kan snakke med noen av lærerne som har mer med overleaf.

Hvor skal vi skrive om bruken av module pattern?

Module pattern er mest i implementasjon. Kan referere fra design delen ned til implementasjon.

Hvordan minimere kodesnutter?

Bruk ... med // comment, for å forkaler hva som blir gjort i delen.

Max lengde av kodesnutt?

Blir kanskje litt mye når deg går over 20 linjer. Kan være det blir lengre noen ganger.

Oppset av neste møte

Onsdag kl 10.15.

Meeting with Supervisor

Participants

Convener: Kolloen, Øivind

Facilitator: Holt, Kent Wincent

Recorder: Torkelsen, Eldar Hauge

Others

Absent: Butt, Zohaib

Agenda

- Feedback on report draft

Minutes

Overordnet struktur ser veldig bra ut. Språket iforhold til å være akademisk er greit men det er noen språk blomster. Noe av det går på korrektur. 1.2 tredje avsnitt, var rett og slett vanskelig å lese. også siste avsnitt under 1.2. siste setning i 1.3 er også litt tung å forstå hva som er meningen. Den er grei men unødvendig tung. Prøv å finne et annet ord for "unprofessional". 1.5 midt i avsnittet, setning med "more theorethical understanding of" prøv å omskrive.

Når en nevner tidsbegrensing; ta med start tid, ikke bare slutt tid. Opperasjonelle krav er veldig vanskelig å måle, må endres.

risiko analyse: må få riktig ordbruk for å vise hva som er sikkerhet i løsningen og hva som er sikkerhet i prosessen. Første krav nevner sikkerhetskrav, det er dette som er problemet.

Ta sikkerhets aspectene ut av risk assessment.

3.1 arkitektur. første avsnitt nevner veldig mye,

første avsnitt under backend er også veldig vanskelig å lese

under figur scrum board for test sprint, sier delt i fire når figuren viser 5 deler (Done).

Ser ikke ut til å være noen feil med forholdet mellom implementation og development

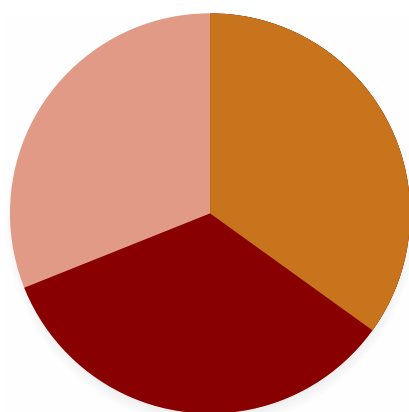
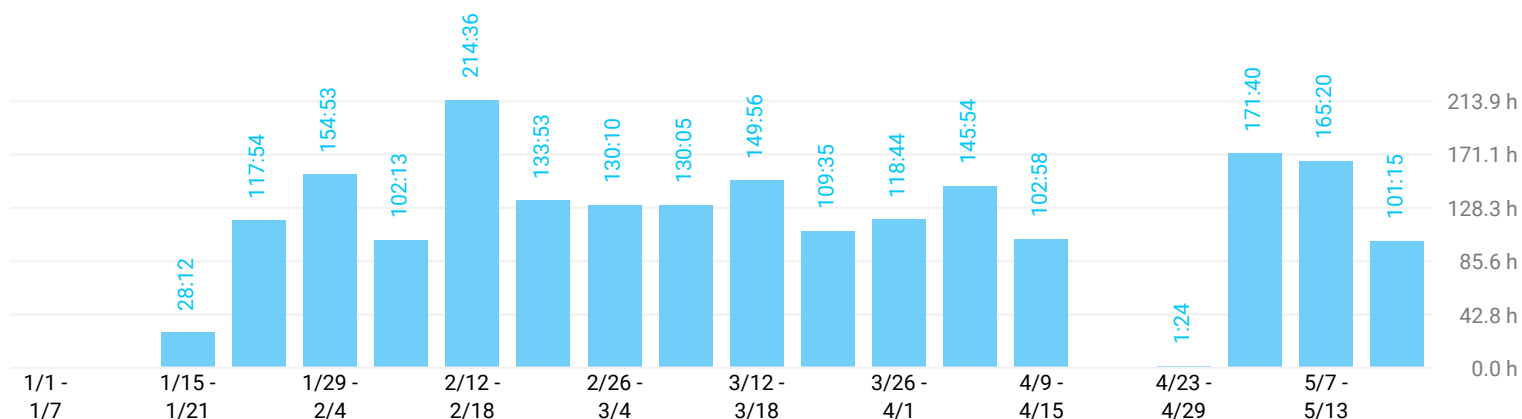
deployment bør være etter implementasjon når det er deploy på hardware.

H Time log

Summary Report

January 01, 2019 – May 17, 2019

TOTAL HOURS: 2078:49:04

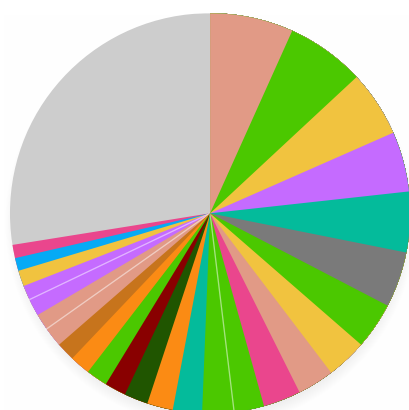


USER

- EL Eldarht
- ZO Zohaibb
- KE Kentwh

DURATION

726:21:32
707:04:05
645:23:27



TIME ENTRY

- Report writing
- Thesis writing
- Bachelor thesis
- Project plan
- COD-8: As a user I would like to see what functions are tightly coupled in the 3D representation
- Without description
- COD-47: Get what functions are called by other function
- COD-57: Show things belonging in a scope inside its scope in the visualization
- Project Plan
- COD-56: As a user i would like to see classes parsed for both java and cpp target
- COD-27: Frontend refactoring
- COD-11: As a user i would like to view or hide the implementation of a function or class by clicking it.
- COD-19: As a user i would like the data structures in the 3D visualization to group together with related structures
- Pair programming
- COD-72: Write tests for java Models


DURATION

139:06:01
131:40:22
110:30:12
103:23:01
100:33:25
92:05:04
79:41:40
68:12:33
62:06:54
60:42:40
52:57:27
50:29:50
50:17:13
42:03:14
39:54:17

● Technological research	37:43:24
● COD-3: As a user i would like to see a representation of a simple function	35:32:49
● COD-4	32:59:37
● COD-62: Write tests for model/controller repo.go	32:28:44
● COD-17: As a user i would like to see what namespace datastructures are in.	32:11:32
● COD-6: As a user i would like to submitt a git repository link for the system.	28:17:01
● COD-49: Add websocket for initial request backend	27:45:21
● Pair programming - COD-57: Show things belonging in a scope inside its scope in the visualization	25:34:07
● COD-28: Setup imgui	25:09:10
● COD-71: Write tests for Cpp and Java Lstnr_initial.java	22:43:34
● COD-60: Write tests for CodeSnippetController	21:00:43
● Other time entries	573:39:09

USER - TIME ENTRY


DURATION

 Eldarht	726:21:32
cache	0:57:22
COD-17: As a user i would like to see what namespace datastructures are in.	32:11:32
COD-27: Frontend refactoring	19:43:42
COD-29: Find a way for antlr to parse only scope.	7:39:31
COD-34: Setup frontpage for userstudies	19:31:42
COD-34: Setup frontpage with information about the project	9:09:30
COD-38: Add websocket for initial request	14:34:36
COD-3: As a user i would like to see a representation of a simple function	2:47:23
COD-41: Add api endpoint for listing saved repositories	0:00:10
COD-42: Make antlr scope sensitive	9:04:00

USER - TIME ENTRY

DURATION

COD-44: Parse number of classes in a file	4:32:59
COD-45: Parse lines of code in a file	8:38:22
COD-47: Get what functions are called by other function	1:03:12
COD-48: Add logger for error login on backend	19:41:00
COD-49: Add websocket for initial request backend	27:45:21
COD-53: Research cyclometric complexity quality metric and find existing solutions	4:54:19
COD-54: Research connascence quality metrics and find exisiting solutions	2:55:32
COD-56: As a user i would like to see classes parsed for both java and cpp target	12:03:24
COD-57: Show things belonging in a scope inside its scope in the visualization	68:12:33
COD-60: Write tests for CodeSnippetController	21:00:43
COD-62: Write tests for model/controller repo.go	32:28:44
COD-66: Write ava test for repo/list	5:11:26
COD-6: As a user i would like to submitt a git repository link for the system.	28:17:01
COD-8: As a user I would like to see what functions are tightly coupled in the 3D representation	39:31:19
Configuring ip/domain	1:20:00
Docker	6:38:37
Docker-compose	3:53:00

USER - TIME ENTRY	DURATION
Jenkins/OpenStack research	15:24:51
Meeting with supervisor and product owner.	1:18:42
OpenStack setup	7:48:32
Project plan	45:11:57
Sprint planning meeting and meeting planning meeting	0:30:06
status report	7:23:04
Thesis writing	131:40:22
User studies	4:00:04
Web architecture	9:14:13
webgl	10:00:33
Without description	90:02:08
 Kentwh	645:23:27
Backend architecture research	2:06:00
COD-19: As a user i would like the data structures in the 3D visualization to group together with related structures	50:17:13
COD-27: Frontend refactoring	33:13:45
COD-3: As a user i would like to see a representation of a simple function	32:45:26
COD-43: Parse number of functions in a file	8:32:27

USER - TIME ENTRY


DURATION

COD-45: Parse lines of code in a file	4:51:23
COD-50: Add websocket for initial request frontend	17:59:50
COD-56: As a user i would like to see classes parsed for both java and cpp target	48:39:16
COD-5: As a user i would like to be notified if my browser does not support WebGL or this application	11:59:41
COD-71: Write tests for Cpp and Java Lstnr_initial.java	22:43:34
COD-72: Write tests for java Models	39:54:17
COD-8: As a user I would like to see what functions are tightly coupled in the 3D representation	6:22:27
ImGUI integration test with THREE.js	3:50:51
Launching development instance	3:19:00
Meeting	15:16:37
Meeting about + with Kolloen	0:47:01
Meeting with Christopher	6:29:03
Meeting with Frantz	4:10:26
Meeting with Kolloen	4:33:02
Minor code refactoring	4:10:37
Pair programming	42:03:14
Planning for meeting with Kolloen	0:17:54

USER - TIME ENTRY

DURATION

Professional programming report writing	7:10:56
Project plan	58:11:04
Report correction	14:59:11
Report reading	9:29:26
Report writing	139:06:01
Research	10:51:19
Sprint planning meeting	0:19:00
Sprint retrospective	1:21:21
Technological research	26:45:44
User studies	3:13:21
WebGL Workshop	9:33:00

 Zohaibb	707:04:05
Bachelor thesis	110:30:12
cache	2:19:20
COD-11: As a user i would like to view or hide the implementation of a function or class by clicking it.	50:29:50
COD-25: As a developer i would like the API to include request for implementation based on function/class/etc name or start/end markers sent with other API endpoint	8:23:49
COD-28: Setup imgui	25:09:10

USER - TIME ENTRY

DURATION

COD-30: Setup window for displaying code to show on response from backend	2:18:53
COD-31: Send request to api when click on 3Dview function object.	10:36:59
COD-32: Update function response	0:50:39
COD-39: Add websocket for initial request frontend	11:20:22
COD-4	32:59:37
COD-40: Add IMGUI fileds for displaying available repositories	6:28:29
COD-47: Get what functions are called by other function	78:38:28
COD-56 pair programming	1:41:39
COD-57 pair programming	7:55:41
COD-6	7:03:19
COD-60 pair programming	0:28:00
COD-63: Write tests for validation.go	3:12:37
COD-64: Write tests for typeLogger.go	9:18:30
COD-65: Write ava test for repo/add	18:41:51
COD-69: Write tests for Cpp and Java ParserFacade.java	15:32:03
COD-70: Write tests for Cpp and Java ExtendedListener.java	2:05:00
COD-73: Write test for MongoDB.go	8:18:44

USER - TIME ENTRY

DURATION

Cod-8 review	0:01:24
COD-8: As a user I would like to see what functions are tightly coupled in the 3D representation	54:39:39
Configuration DevOps	7:55:43
Decisions	4:49:10
dockerization	7:55:35
Internal meeting	4:17:04
Kolloen meeting	0:43:37
Meeting with Christopher	10:15:20
meeting with Kolloen	1:50:08
Pair programming - COD-56: As a user i would like to see classes parsed for both java and cpp target	17:22:03
Pair programming - COD-57: Show things belonging in a scope inside its scope in the visualization	25:34:07
Planning meeting	1:30:47
Planning meeting + meeting with Kolloen	1:25:47
Project Plan	62:06:54
Project Plan + Kolloen meeting	0:47:30
Project setup	3:12:54
Raport correction	11:56:28

USER - TIME ENTRY

DURATION

Reading thesis	4:22:55
server configuration	6:10:09
Sprint retrospective meeting	17:57:39
Status rapport writting	6:50:56
Technological research	10:57:40
Trying out antlr with goLang	11:06:03
Trying out antlr with java	2:50:44
User studies	3:53:41
WebGL workshop	10:04:00
Without description	2:02:56