Tampere University Unit of Computing Sciences

TIEA4 Project Work (City centre campus)

TIETS19 Software Project Management Practice (City centre campus)

Software component manager

Final Report

|  |
| --- |
| K424990: Jyry Uitto |
| K431258: Petri Nieminen  K427770: Jani Aakio  K439656: Kaisa Sormunen  K427803: Pekka Oinas  K427534: Sanni Onkalo  K439136: Denis Kole |

Version history

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 0.1 | 20.05.2021 | Jyry Uitto | First draft |
| 0.2 | 23.05.2021 | Jyry Uitto | Done for peer review and finishing touch. |
| 1.0 | 24.5.2021 | Jani Aakio | Reviewed and small touches |
|  |  |  |  |

Contents

1 Introduction 4

1.1 Purpose of the report 4

1.2 Product and environment 4

1.3 Definitions, abbreviations and acronyms 4

2 Project organisation 4

2.1 Group 4

2.2 Customer 5

2.3 Other stakeholders 5

3 Project implementation 5

3.1 Communication 5

3.2 Tools and technologies 6

3.3 Sprints 6

3.4 Deliverables and outcomes 8

3.5 Restrictions and limitations 9

3.6 Third party components, licenses and IPRs 9

4 Working hours 9

5 Quality assurance 13

5.1 General description of testing 13

5.2 Bug reporting 13

5.3 Conclusions on product’s quality 14

6 Risks and problems 14

6.1 Foreseen risks 14

6.2 Risks not foreseen 14

6.3 Communication challenges 14

6.3.1 Communication practices and tools 14

6.3.2 Communication related challenges 14

6.3.3 Strategies and solutions 15

7 Not implemented in this project 15

7.1 Rejected ideas 15

7.2 Further development 15

8 Lessons learnt 15

9 Comments about the course 15

10 Statistics 16

# Introduction

## Purpose of the report

This document gives details about the software component manager development process. It gives information about tools that were used to develop the product, lessons learned from developing this product. Furthermore, it provides details about the product itself such as Information about possible bugs that were found.

## Product and environment

Software component manager.

Software component manager is a product that helps to maintain information about components used by the user of this product. It provides easily maintainable database of software used, versions inside such software, in which modules these components are used. As an addition the software component manager allows user to compare the different components inserted into the database and see the possible differences between objects.

The product is built on three Docker containers. These three docker containers are UI, Controller, and Database. User will access either UI or controller container to use the product. Controller container uses the database container to store information, and UI container uses Controller container as an intermediary.

# Project organisation

## Group

Project managers:   
Jyry Uitto, Jyry.Uitto@gmail.com 0400602108

Jani Aakio,Jani.Aakio@tuni.fi 0408412601

Developers:

Petri Nieminen

Kaisa Sormunen   
Pekka Oinas

Sanni Onkalo

Denis Kole

## Customer

Accenture

Juha Tuominen, 0405040774, [juha.t.tuominen@accenture.com](mailto:juha.t.tuominen@accenture.com)

## Other stakeholders

Tampere University.

Pekka Mäkiaho, 0505566266, Project supervisor

Timo Poranen, Course coordinator

# Project implementation

## Communication

Communication was arranged via Teams meetings.

The team itself met weekly on Wednesdays to maintain project status.

In the beginning of the project there were a couple of customer meetings to get the required requirements and the project started properly.

The stakeholders were kept informed about the project progress via review meetings that occurred ~once a month.

Used tools to communicate between team were Discord and Teams.

Discord handled daily communications and Teams was there to arranged meetings.

## Tools and technologies

Tools and technologies demanded to be used:

Docker: Provides environment to run the product that runs properly with any system.

Docker-Compose: Eases the use of multiple docker containers, by providing tools to use them.

Scala: Product logic language, handles all interaction with the product.

Angular: Product User interface

Nginx: Server that runs the Angular UI.

Postgres: Database of the product.

Self-chosen tools and technologies.

* Oracle Virtual machine / Linux. This was used to run docker containers without issues.
* Github: Version control.
* Intellij Idea or any other Integrated development environment.
* MMT: Reporting for University stakeholders, also to keep track on some project status statistics.
* Python: Tests are written with python because the team had previous knowledge.
* OpenApi: Specification to standardize generated Rest apis.
* Restful api: Specification to generated standardized Rest apis.

## Sprints

9 Sprints with length of 2 weeks.

First sprint was to get tools installed for every member in the team.

All other sprints were used to implement items into the product.

Last sprint had polishing after features were completed.

6 items in to do.

5 items in progress.

48 items are completed.

8 are rejected.

In the sprints the info is marked as TODO/IN PROGRESS/COMPLETED/REJECTED

Sprint 1

0/0/0/0

Sprint 2

8/7/0/0

Sprint backlog = 7 items

Completed = 7 Items

Sprint 3

4/6/7/0

Sprint backlog = 6 items

Completed = 1 Item

Sprint 4

2/8/8/0

Sprint backlog = 8 items

Completed = 4 Items

Sprint 5

2/7/12/0

Sprint backlog = 7 items

Completed = 14 Items

Sprint 6

13/6/26/0

Sprint backlog = 6 items

Completed = 3 Items

Sprint 7

13/9/29/0

Sprint backlog = 9 items

Completed = 4 Items

Sprint 8

12/8/33/0

Sprint backlog = 8 items

Completed = 10 Items

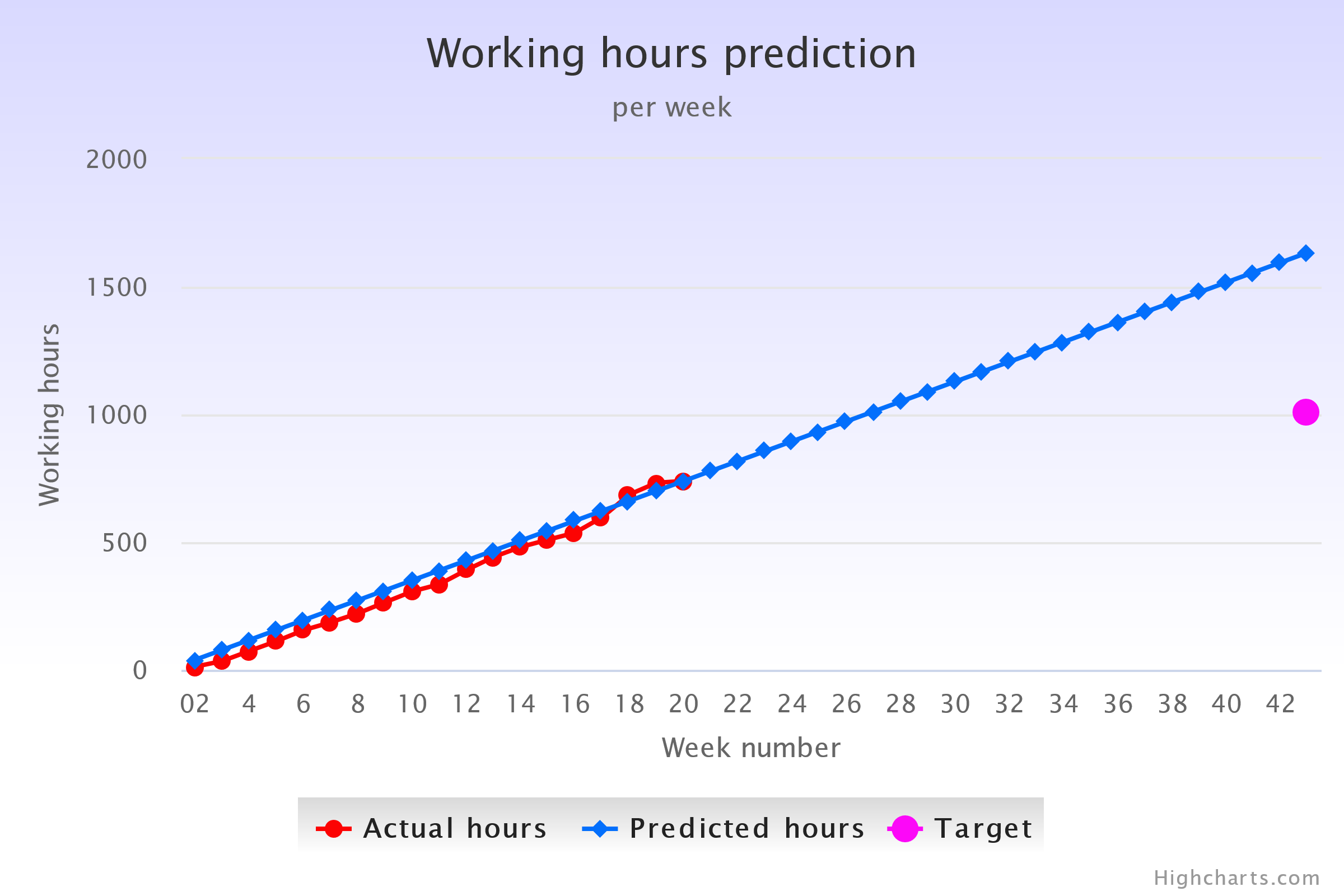
Sprint 9

9/3/43/8

Sprint backlog = 3 items

Completed = ? Items

Prioritization was quite successful. All relevant components are delivered and the product itself is complete. Security was scrapped as it became apparent that it will not fit the scope of the project.



## Deliverables and outcomes

Product will be delivered via Github repository link:

<https://github.com/Bunnyface/sw_comp_project>

It contains everything needed to build the service images. We believe the customer uses ansible to set-up their docker so it is left to the customer to take these images into use.

Under is mostly a copy from the readme from the product desribing it:

### General information

Software component manager is a product designed to help with overseeing modules and components and their versioning.

The overall service is based on a simple Model-View-Controller architecture where each is represented by a distinct container.

The product is composed of three distinct services:

* UI/View is an Angular based nginx container. It offers a UI to use with the product.
* Controller is a Scala container based on an openjdk image. It uses hseeberger/scala-sbt image meant to be used for Scala-based services.
* Model is an official postgres container and not much custom work has been done on this image.

### Container information:

#### Angular container

The folder angular contains the un-compiled version of angular service. By using the docker file in this folder, it will be compiled Ahead-of-time and the service built from it will be copied to the folder angular\_dev. This is not fully necessary when building the service, but helps with de-bugging and quickly re-building the service.

To make the service truly workable with docker, it must be copied into a server cabable of running it. This product uses a nginx-docker image. The base-image for this can be found in the folder angular\_dev

After the image has been built, it can be ran using either the docker-compose found within this product, or can be imported to other systems if needed.

If the docker-compose is used, the angular service is available at the address: 0.0.0.0:8081. It servers traditional HTTP server that can be used as a GUI for the rest of the service.

Much of the code within angular folder is generated code. The hand made part is mostly found within /src/app

Each folder describes pretty well what it contains:

* Compare: Comparing different modules
* Components: Displaying components
* Data input: Inputting data into the database
* Insert-comp-to-mod: Dealing with component to module relation inputs
* Insert-component: Inserting components
* Insert-file: Inserting Json-files to the database
* Insert-module: Inserting modules to database
* Release-detail: Getting info on particular module
* Releases: Getting all modules
* Shared: Code shared by multiple parts of the code

#### Scala container

The Scala based container serves as a controller container. It serves a REST API that offers a simple way to interact with the database of the product.

The API has both open-ended and more strict endpoints when it comes to the payload that they expect.

The files needed to build this image are in the folder *scala*.

Simple breakdown of the .scala files:

* Main: The main body of the scala. HTTP service is started here and the endpoints
* Models: The models used when inserting to database
* Client: SQL client used when connecting to the database
* Retrieve: The functions used to pull data from the database
* Send: The functions used to insert data into the database
* Delete: The functions used to delete data from the database
* Comparison: The functions used to compare different models from the database

#### Postgres container

Not much is to be said of this container. It is a pre-built container which has very little customizing done on it as it works as advertised without much work.

### Others:

There is also a tests folder which contains the integration tests. These are ran with a python container. The purpose of these is to test the actual API endpoints and see that the results that we get from those endpoints are what we expect. This is our main way of testing the product because it shows what happens when you actually run requests to the container. As such it is the best way to make sure that the service does what it’s expected to do.

### Additional information

Total amount of source code: Approx. 25 000 lines of code. (Excluding the fully generated angular\_dev folder)

Self-coded lines: Approx. 3000 lines. The accuracy of this might be a little off as some files have generated code within them.

Number of main-views: 3

Number of Scala classes: 8 (Strict structures were omitted)

Number of Scala functions: depending on how you count, minimum of 57 functions.

## Restrictions and limitations

Security with Wso2 was dropped.

## Third party components, licenses and IPRs

Accenture has all the rights to do with the product as they wish.

Scala image is based on apache licence according to its page, but even that might not be completely true.

As docker files are usually whole OS’s, their licences are somewhat tricky. The ones images pulled from the docker registry, should all be open-source, but because they are whole OS’s, they usually fall under multiple licences.

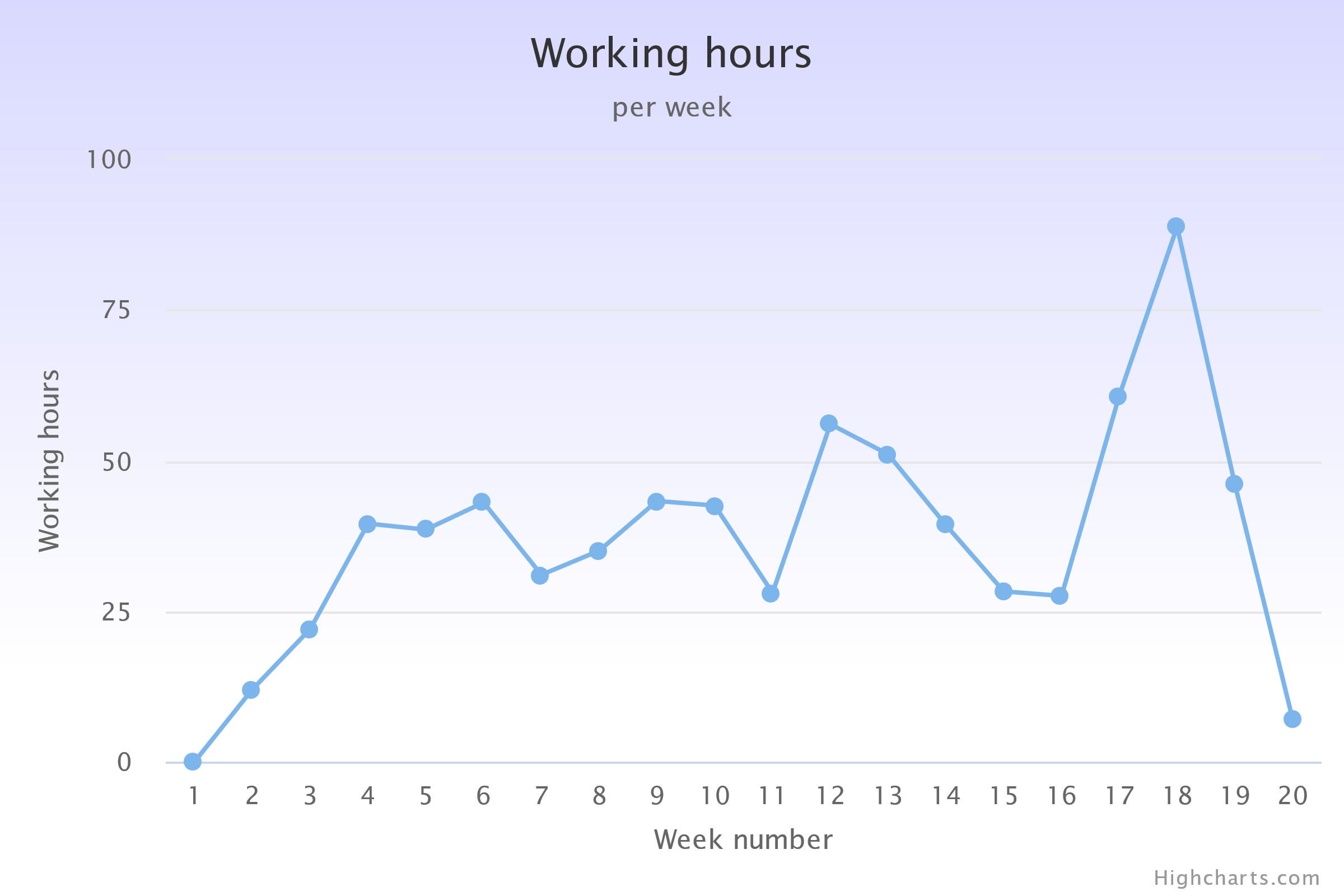
We as a team do not have the resources to check every part of these images and check under what licences they fall. Unfortunately, this must be left to the customers discretion on what base images to use.

# Working hours

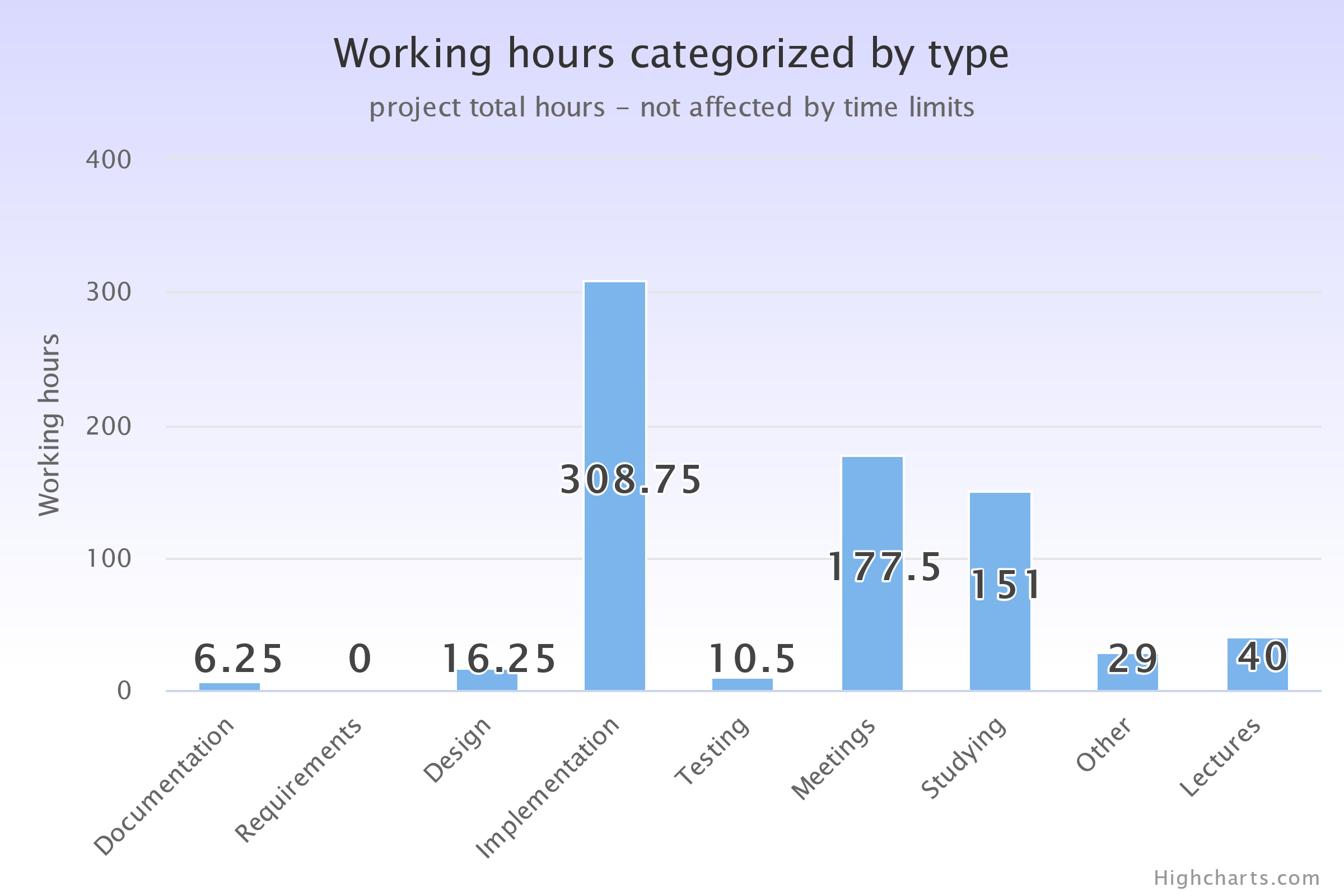
Up to date in morning of 24.5.2021

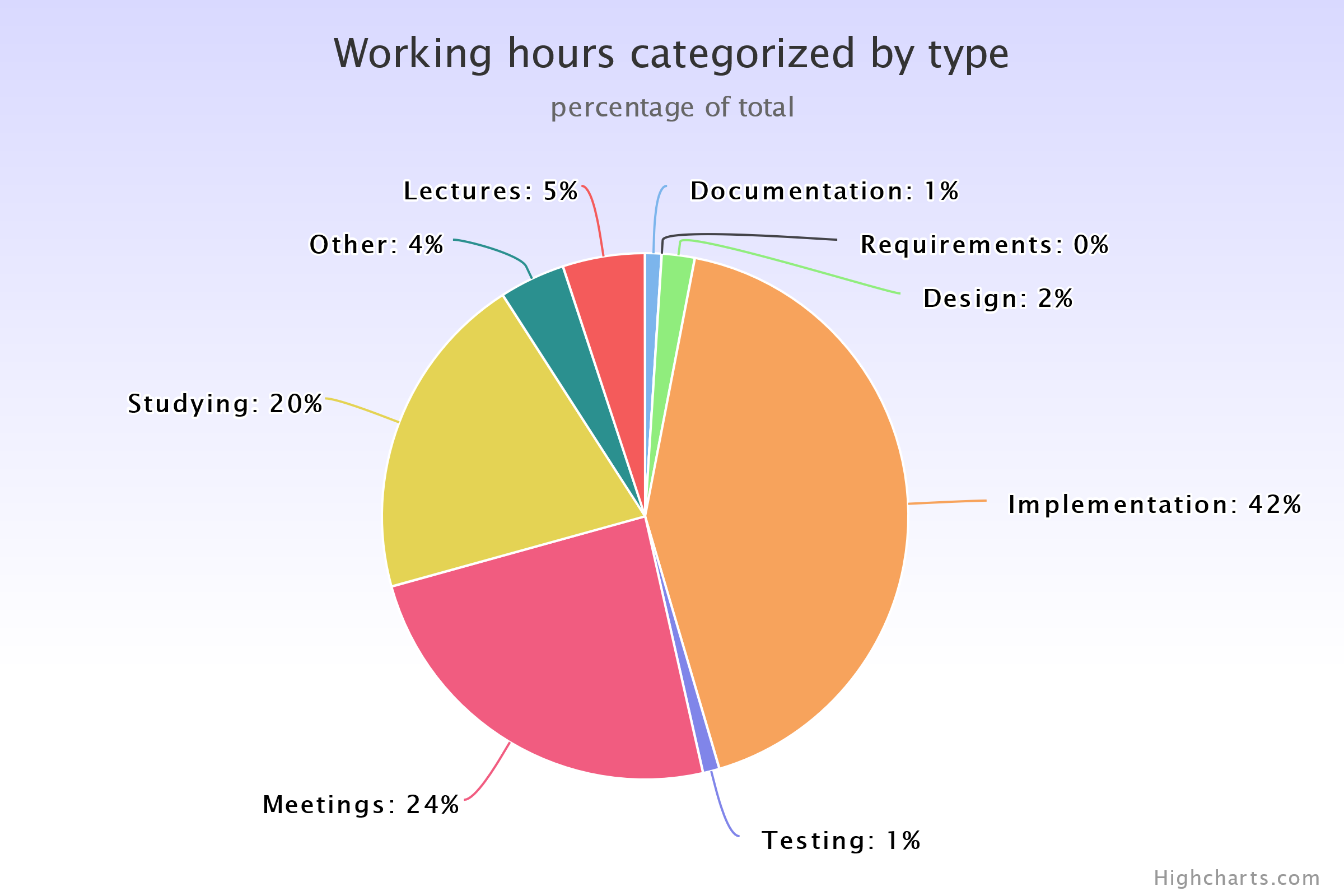
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Jyry | Jani | Petri | Kaisa | Pekka | Sanni | Denis |
| Documentation | 5.75 | 13.5 | 0 | 0 | 0 | 0 | 0 |
| Requirements | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Design | 2.75 | 3 | 9.5 | 0 | 0 | 1 | 0 |
| Implementation | 27.5 | 67.5 | 2.5 | 54 | 20.5 | 67.25 | 77.5 |
| Testing | 20 | 9.5 | 0 | 3 | 0 | 1 | 0 |
| Meetings | 32.5 | 35.5 | 23.5 | 24 | 14 | 26.25 | 21.75 |
| Studying | 73.25 | 1 | 10 | 18 | 3 | 24.75 | 21 |
| Other | 9.25 | 4 | 0 | 8 | 2 | 5.75 | 0 |
| Lectures | 10 | 9 | 0 | 6 | 4 | 7 | 4 |
| Total | 161 | 143 | 45.5 | 113 | 43.5 | 133 | 124.25 |

|  | Weekly total |
| --- | --- |
| Week 1 | 0 |
| Week 2 | 12 |
| Week 3 | 34 |
| Week 4 | 74 |
| Week 5 | 112 |
| Week 6 | 155 |
| Week 7 | 186 |
| Week 8 | 221 |
| Week 9 | 264 |
| Week 10 | 307 |
| Week 11 | 335 |
| Week 12 | 381 |
| Week 13 | 442 |
| Week 14 | 481 |
| Week 15 | 509 |
| Week 16 | 537 |
| Week 17 | 598 |
| Week 18 | 686 |
| Week 19 | 732 |
| Week 20 | 739 |
| Total | 739 |









# Quality assurance

## General description of testing

The project has self-built tests available in GitHub repository. They were used to check that a new version of the product does not break previously created features and would pass all the tests made for the product. Tests applied mock data to test individual components.

Tests were created by Jani, Kaisa, Pekka, Dennis and Sanni. They were used as a continuous integration loop by all developers.

## Bug reporting

The product appears to be bug-free. Due to the nature of the service. The scala container which might have the most possible bugs, should be resistant to failure. If an endpoint does give a failure. It should be caught and sent as a internal-server error. This means that the product has fault tolerance even in cases where bugs might occur.

## Conclusions on product’s quality

Product does what it is supposed to do and does so without any apparent problems.

It passes all self-made tests and so it should be stable.

Security features were dropped. Thus overall grade is good, but additional features and properties would improve it to great.

# Risks and problems

## Foreseen risks

|  |  |  |  |
| --- | --- | --- | --- |
| Risk ID | Explanation, severity/impact, probability, size/importance | Impact/Severity | Probability |
| 2 | Members lack motivation/time to do the project | High | Low |
| 4 | Lack of required skills | Medium | High |
| 5 | Hardware problems | Medium | Low |
| 6 | Not well-defined requirements | High | Very Low |

## Risks not foreseen

Communication is not instant and there were some time periods where we had to wait for answers from the client.

## Communication challenges

This years remote working definitely caused some problems. It is hard to form any sort of team spirit or bonding when all team-members haven’t even seen each others faces. It’s also much harder to get a good feedback loop going when everyone feels a bit distant from eachother.

## Communication practices and tools

Discord was used to everyday chatting.

Teams was used to hold meetings.

Teams task board was used to keep every member’s current project status updated.

Sharing desktop view to showcase and tutor other members to new things.

## Communication related challenges

Different people had different timetables, so working together at some aspects of the project was not easy.

## Strategies and solutions

We used weekly meetings to update status of the project and assign new tasks.

# Not implemented in this project

## Further development

Wso2 security. Client suggested this feature, but it was dropped since it exceeded projects scope.

# Lessons learnt

There was a lot of tools that one had to learn how to do the basics with, some developers got into more advanced stuff in their chosen technologies and languages.

Task sharing went well. It was really straightforward to assign tasks to people, since everyone had a thing they wanted to do and the overlap was small, so everything was being handled. Beforehand I thought that this will be far more troublesome.

Overall the project went really smoothly, and no major hiccups happened. Yet for the next project there is room for improvement in:

### Project management:

* Keeping the team motivation up and making sure that everyone is getting enough work done.
* Keep up to pace with the requirements etc. Sometimes its easy to see in your own head what needs to be done but communicating that is difficult

### Team forming:

* Team was quite individualistic and doing their own things, some workshops / Arranged work schedule for teamworking could have been nice.
* Remote working is definitely more difficult when dealing with teams. When its impossible to do group programming etc. Things become much harder

# Comments about the course

The project managing basics part of the course should be done prior to the course. Getting insight into project management after the project is finished feels wonky.

# Statistics

Software component manager

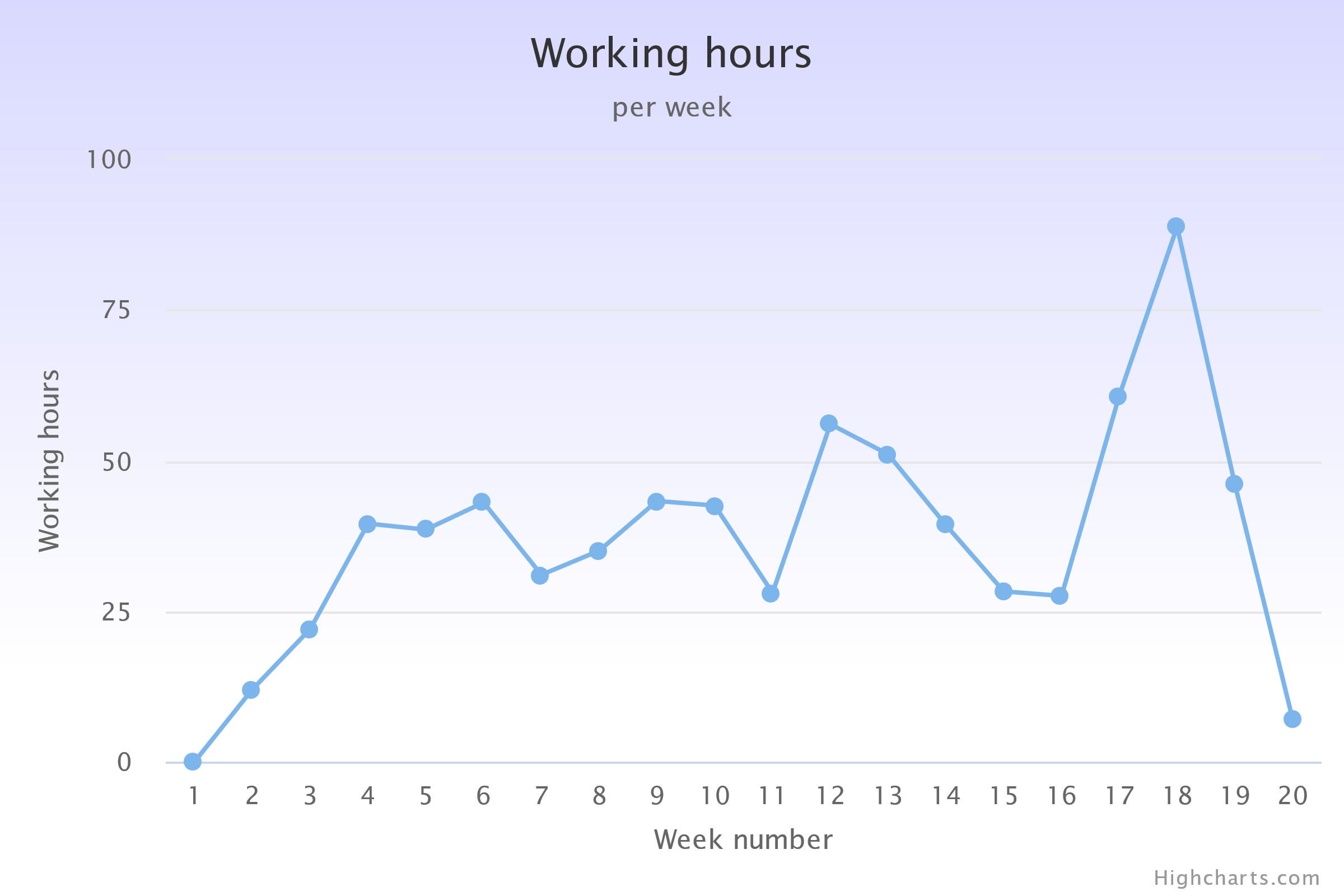
7 Developer team.

* Tools and technologies that were used:
* Docker.
* Docker-Compose.
* Scala.
* Angular.
* Nginx.
* Postgres.
* Oracle Virtual machine -> Linux.
* Github.
* Intellij Idea.
* MMT.
* Python.
* OpenApi.
* Restful api.

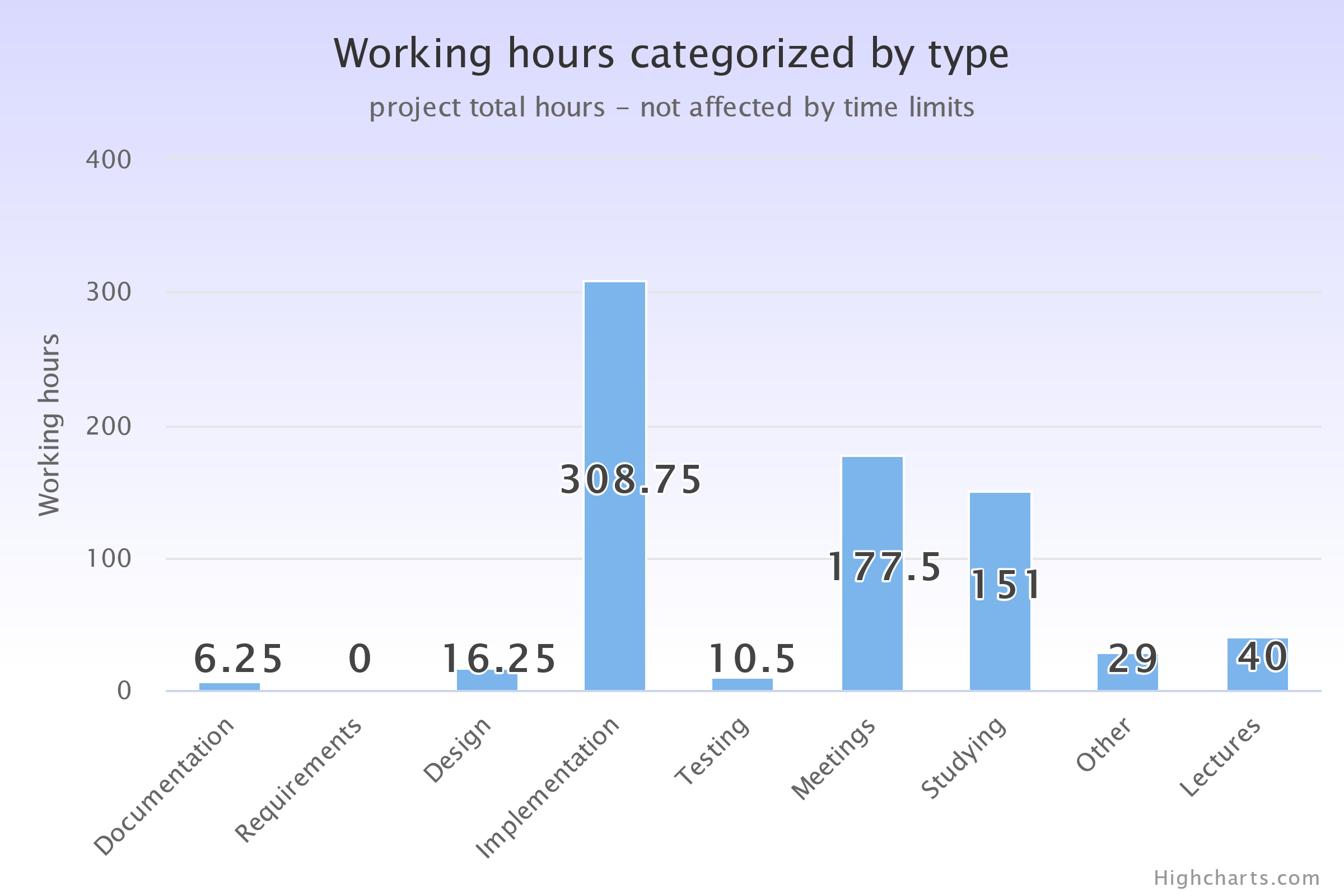
------ Lines of code etc ------

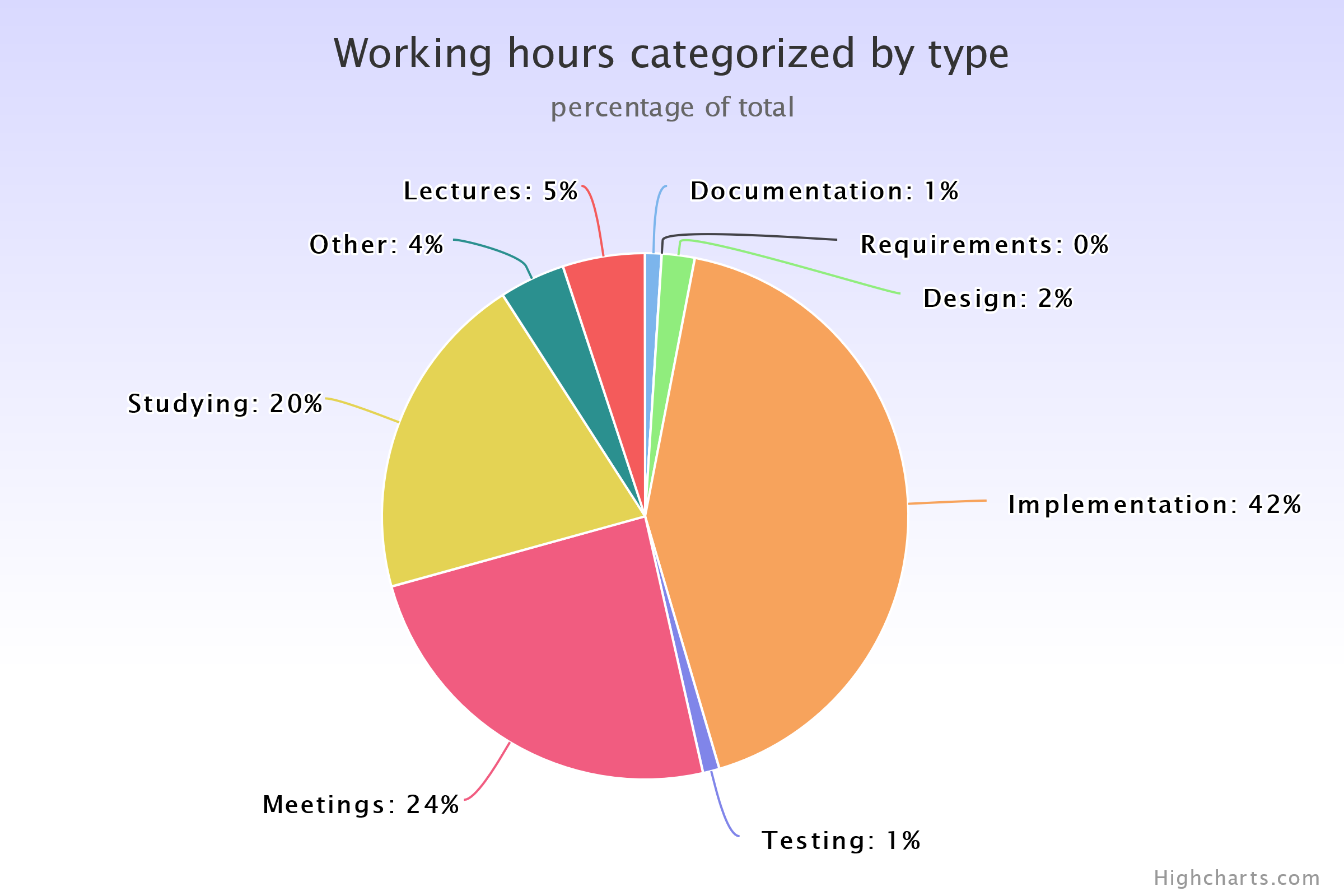
------ Methods / Classes ------

246 version control commits









43 implemented items, 8 rejected and 4 still in backlog.