University of Zurich^UZH

Department of Informatics

Binzmühlestrasse 14
CH-8050 Zürich-Oerlikon
Switzerland

Prof. Dr. Harald C. Gall
Software Evolution and
Architecture Lab

Phone +41 44 635 43 35
Fax +41 44 635 68 09
gall@ifi.uzh.ch
http://seal.ifi.uzh.ch

Timothy Zemp
Hagenholzstrasse 92
8052 Zürich

Matrikel-Nr. 13-915-632
timothy.zemp@uzh.ch

July 24, 2020

Master's Thesis Specification

Separating wheat from chaff: Detecting genuine Continuous Integration configurations

Continuous Integration (CI) is a software development practice introduced by the Agile movement with the aim of delivering reliable software releases fastly [3]. Whenever a change is pushed in a remote software repository (*e.g.*, GitHub), a build server tool (*e.g.*, Travis-CI) detects this change and triggers the integration pipeline. The pipeline typically consists of several steps *e.g.*, compilation, unit testing, packaging, integration testing, installation and deploy and is usually defined within a configuration file (*e.g.*,.travis-ci.yml).

CI has also been an uprising topic in empirical research, studying the impact of CI on code review or developer's behaviour. Notable studies showed that for example CI leads to a higher code commitment frequency and reduced code chunks [6], different programming languages showes different adoption times (*e.g.*, Ruby as being an early-adopter, whereas Java projects tend to adopt later) [5], and proper build configurations should be maintained in order to reduce long build durations [2]. Cloud CI providers (*e.g.*, Travis-CI, Jenkins) are easy to adopt and once a repository is registered, it is sufficient to commit a configuration file into the repository. Due to this simple process, many public repositories contain such configuration files, but naturally, many of them do not use CI seriously. Unfortunately, many studies do not differentiate and typically treat every project with a configuration file as a valid instance (*e.g.*, [6], [5], [2]), which might dilute the results.

Inspired by the work of Munaiah et al. [4], where they implemented and validated a tool (reaper[1]) to asses the quality of a GitHub repository, we aim to lower the tread to validity of empirical studies on CI by providing a classifier which evaluate the *genuineness* of pipeline configurations. Reaper provides a score which assesses the likelihood of the repository containing engineered software separating it from potential noise (*e.g.*, home work assignments, study projects, ...). The tool works by identifying different practices, called dimensions (*e.g.*, documentation, testing, history, ...) and classify them using both a Score-Based Classifier as well as a Random Forst Classifier. By establishing a Ground Truth Dataset and extracting features on the pipeline configuration, we aim to assess the quality of CI within a software repository.

[1]https://github.com/RepoReapers/reaper

The goals of this Master's thesis

The goal of this master's thesis is to identify genuine configurations that qualitative CI typically express with the intention to generalize those in order to identify software repositories or configuration adopting these practices. Such a classifier (similar to reaper) would either accepts an CI configuration file or a software repository and outputs a score on the quality of the CI within the software repository. This will allow future studies to filter out software projects that do not have genuine CI configurations, picking only those who uses CI at a high level of quality.

Whilst developing the classifier, the following research questions arise:

RQ 1: Why does software projects have pseudo configuration files?
First we analyze why software projects might have pseudo configurations (*i.e.*, in opposite to genuine configurations). This will enable us to extract features of genuine CI configurations.

RQ 2: What is the accuracy (Precision & Recall) of an automated detection of meaningful CI configurations?
In RQ 2 we focus on measuring the accuracy of our classifier. By segment-comparing (approx. 10%) of our manually labeled reference-set we calculate precision (false positive rate) and recall (false negative rate) and hence measure the accuracy of the classifier.

RQ 3: Are features extract only from the configuration sufficient to classify accurately or is a more broader context required?
The features of our classifier can be grouped into three categories, each representing a different level (*i.e.*, higher & more difficult) on how those features are obtained: *configuration-level*, *repository-level* and *pipeline-level*. In RQ 3 we analyze whether the lowest level of access (configuration-level) is sufficient to yield accurate classifications, or more broader features (repository-level or pipeline-level) are required.

RQ 4: Does the classification obtained by *reaper* correlate with our classification?
RQ 4 aims to reveal whether there is a correlation between engineered software projects, classified by *reaper* and software repositories having a genuine CI configuration (classified by our classifier).

RQ 5: What impact does the classifier has on CI-based studies?
In the last research question we want to asses the impact of the classifier to already existing empirical CI-studies. Using a large-scale dataset we aim to reproduce previous studies where one for example has analyzed the impact of the programming language towards the adoption of CI, or the impact of CI on the software's quality, but applying our classifier to the selection of the analyzed projects.

Task description

The main tasks of this thesis are:

ID	Task
T1	Research on the topic of CI and the different pipeline configuration mechanism
T2	Data Collection of various repositories using CI and labeling them to establish Ground truth
T3	Definition of extracted features (RQ 1)
T4	Development of classifier
T5	Evaluate the classifier using the reference set and statistical methods (RQ 2/3)
T6	Evaluate correlation between *repaer* and the classifier (RQ 4)
T7	Identify existing CI-studies and evaluate the impact of the classifier (RQ 5)
T8	Write a thesis, comprising the points above

Thesis Structure & Deliverable

The thesis consists of several parts. First, basic research is required in order to gain an overview of the different pipeline providers and the possibilities of CI. Afterwards, a Ground Truth Dataset (D1) is established by collecting various repositories adopting CI and labelling them accordingly. Using the data set, suitable features will be identified (RQ 1) and implemented in the classifier. Once enough features are implemented, the classifier is implemented using a prediction algorithm. We conclude the implementation by wrapping the classifier with a Docker Container (D2). This allows other research to easy use the classifier and reproduce our work. We then evaluate the classifier internally (RQ 2/3), evaluate the correlation towards *reaper* (RQ 4) and finally observe the impact of the classifier on existing empirical CI-based studies (RQ 5). Lastly, the thesis is concluded with an academic report.

A detailed project plan including the milestones is attached to this proposal and can be found on page 5.

ID	Deliverable
D1	Ground Truth Dataset
D2	Classifier
D3	Thesis

Provided resources

The student does not need any specific resources other than the starting literature, which is referenced in this document. A workplace in the s.e.a.l. lab will be at disposal.

General thesis guidelines

The typical rules of academic work must be followed. In Bernstein [1] describes a number of guidelines which must be followed. At the end of the thesis, a final report has to be written. The report should clearly be organized, following the usual academic report structure, and has to be written in English using our s.e.a.l. LaTeX-template. As implementing software is also part of this thesis, State-of-the-Art design, coding, and documentation standards for the software have to be obeyed. The product of the thesis is the Ground Truth Dataset, a working classifier as well as the written report, that summarizes the process of developing the tool and its evaluation.

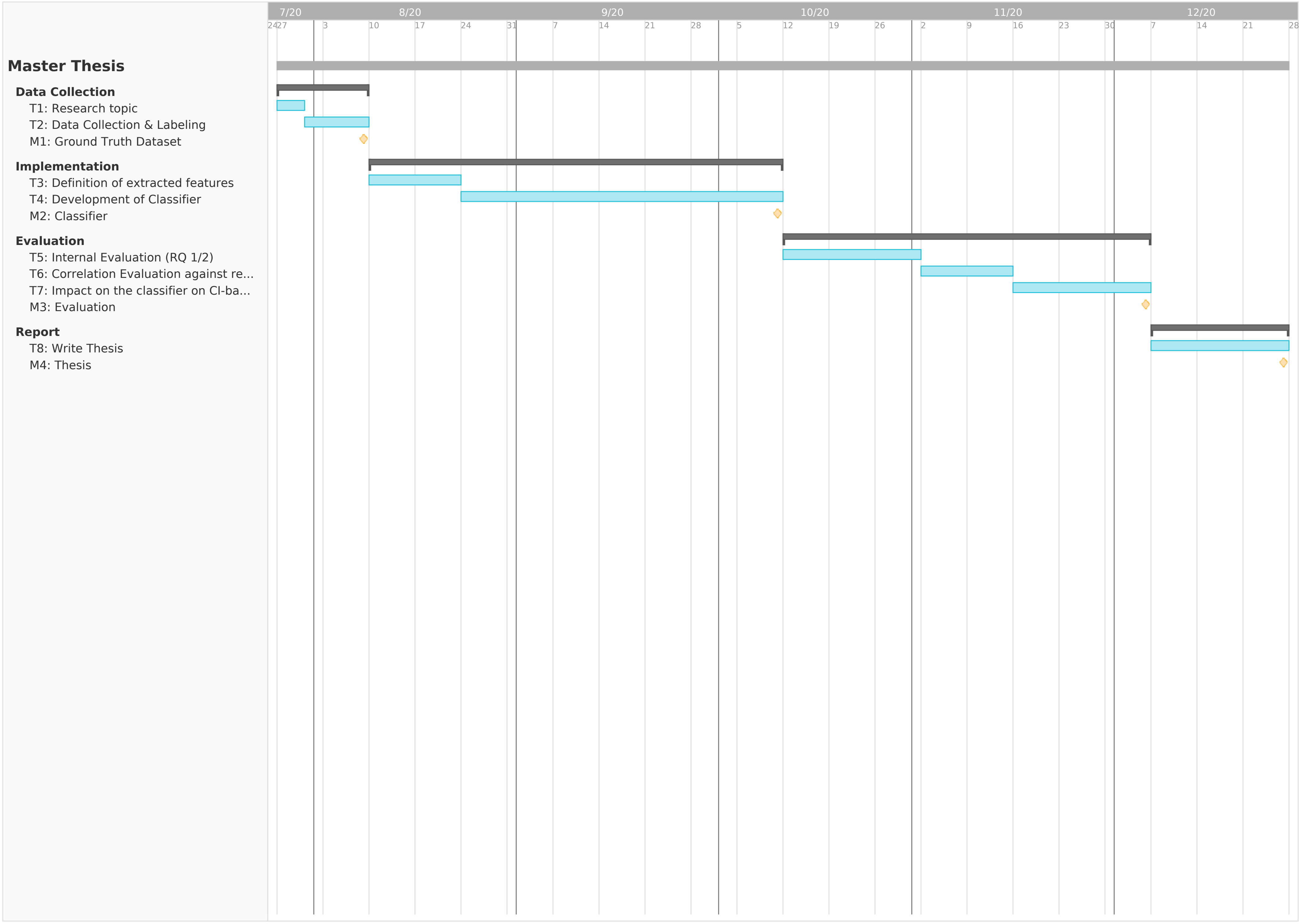
Advisors:

Professor:
Prof. Dr. Harald C. Gall

Responsible assistants:
Dr.-Ing. Sebastian Proksch (TU Delft)

Signatures:

Timothy Zemp   Prof. Dr. Harald C. Gall


Master Thesis
Data Collection
T1: Research topic
T2: Data Collection & Labeling
M1: Ground Truth Dataset
Implementation
T3: Definition of extracted features
T4: Development of Classifier
M2: Classifier
Evaluation
T5: Internal Evaluation (RQ 1/2)
T6: Correlation Evaluation against re...
T7: Impact on the classifier on CI-ba...
M3: Evaluation
Report
T8: Write Thesis
M4: Thesis
7/20
8/20
9/20
10/20
11/20
12/20
24 27 3 10 17 24 31 7 14 21 28 5 12 19 26 2 9 16 23 30 7 14 21 28

References

[1] A. Bernstein. So what is a (diploma) thesis? a few thoughts for first-timers. Technical report, Dynamic and Distribution Information System Group, University of Zurich, 2005.

[2] T. A. Ghaleb, D. A. Da Costa, and Y. Zou. An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, 24(4):2102–2139, 2019.

[3] F. D. Humble, J. Software releases through build, test, and deployment automation. Addison-Weyley Professional, 2010.

[4] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.

[5] B. Vasilescu, S. Van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. van den Brand. Continuous integration in a social-coding world: Empirical evidence from github. In *2014 IEEE international conference on software maintenance and evolution*, pages 401–405. IEEE, 2014.

[6] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 60–71. IEEE, 2017.