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 27, 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 fast [3]. Whenever a change is pushed to 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, *i.e.*, compilation, unit testing, packaging, integration testing, installation and deploy and is usually defined within a configuration file (*e.g.*,.travis-ci.yml in case of Travis CI). The success of CI has also lead to a spike in empirical research on this topic that investigated the impact of CI on code review or developers. Studies show that, for example, CI leads to a higher code commitment frequency and reduced code chunks [6], different programming languages shows 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].

Many free cloud CI providers exist, like Travis-CI or Jenkins, and they are easy to adopt: once a repository is registered, it is sufficient to maintain a single configuration in the repository to use the service. Due to this low entrance barrier, many public repositories contain such configuration files. Naturally, many of them do not use CI seriously though, the developers might have played around with the provider or the repository is a toy-project alltogether. Unfortunately, many studies do not differentiate between these cases and treat every project with a configuration file as a valid instance (*e.g.*, [6], [5], [2]), which might dilute the results, which presents a real threat to validity of an empirical studies on CI.

Inspired by Munaiah et al. [4], who created Rreaper[1], a tool that can detect seriously engineered repositories, we want to do the same for CI configurations and create a tool that can assess the *genuineness* of a pipeline configuration. To this end, we need to identify various *features* that describe the configuration (e.g., length, complexity, or number of configuration changes), build automated extraction facilities, and train a classifier (e.g., Decision Tree, Random Forst Classifier). To assess the success of the approach, we will establish a manually investigated *Ground Truth* dataset that can be used in an automated evaluation.

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

## Research Goals

The goal of this master's thesis is to understand how to automatically detect genuine CI configurations, to study the impact of such a classifier, and to make it easy for future studies to avoid this threat of validity. To reach this goal, it is necessary to answer the following research questions:

**RQ1: Why do repositories contain pseudo configuration files?** First, we analyze the reasons why software projects contain pseudo configurations (*i.e.*, in opposite to genuine configurations). A side effect of this investigation is the creation of a labelled dataset of *genuine* and *pesudo* configuration files.

**RQ2: Is it possible to automatically detect genuine CI configurations?** This RQ provides initial proof that the thesis goal is achievable. We will define an infrastructure that can extract arbitrary CI-related features for existing projects, and use our ground truth dataset to train a classifier and assess its accuracy in a 10-fold cross evalaution.

**RQ3: What is the predictive value of different feature categories?** We intend to extract features on three different levels: *configuration-features* can be extracted from the configuration itself (e.g., how many lines?), *repository-features* can be extracted from the containing repository (e.g., how often has the config been changed?), and *CI-features* can be extracted from the CI provider (e.g., how many builds have been performed in this project?). In this RQ, we will analyze whether simple configuration features are sufficient to yield accurate classifications, or whether advanced feature should be taken into consideration.

**RQ 4: Are *engineered* projects (*reaper*) also *genuine*?** Previous work has identified *engineered* projects. This does not automatically imply a *genuine* CI configuration, but there might be a relation to uncover. We are interested in understanding how both populations of projects compare and whether it is necessary to have two independent classifiers.

**RQ 5: What impact does the classifier have on CI-based studies?** The last RQ presents a quantification for the motivating problem of this thesis. We will replicate simple statistics taken from other empirical studies and compare the results when created on *genuine* and *pseudo* configurations.

In addition to the experimental results, this thesis will produce a usable tool that allows future studies to filter out software projects that do not have genuine CI configurations such they can focus on projects with a meaningful CI configuration.

## Task description

The main tasks of this thesis are:

**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 *reaper* 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 (D3).

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

**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:

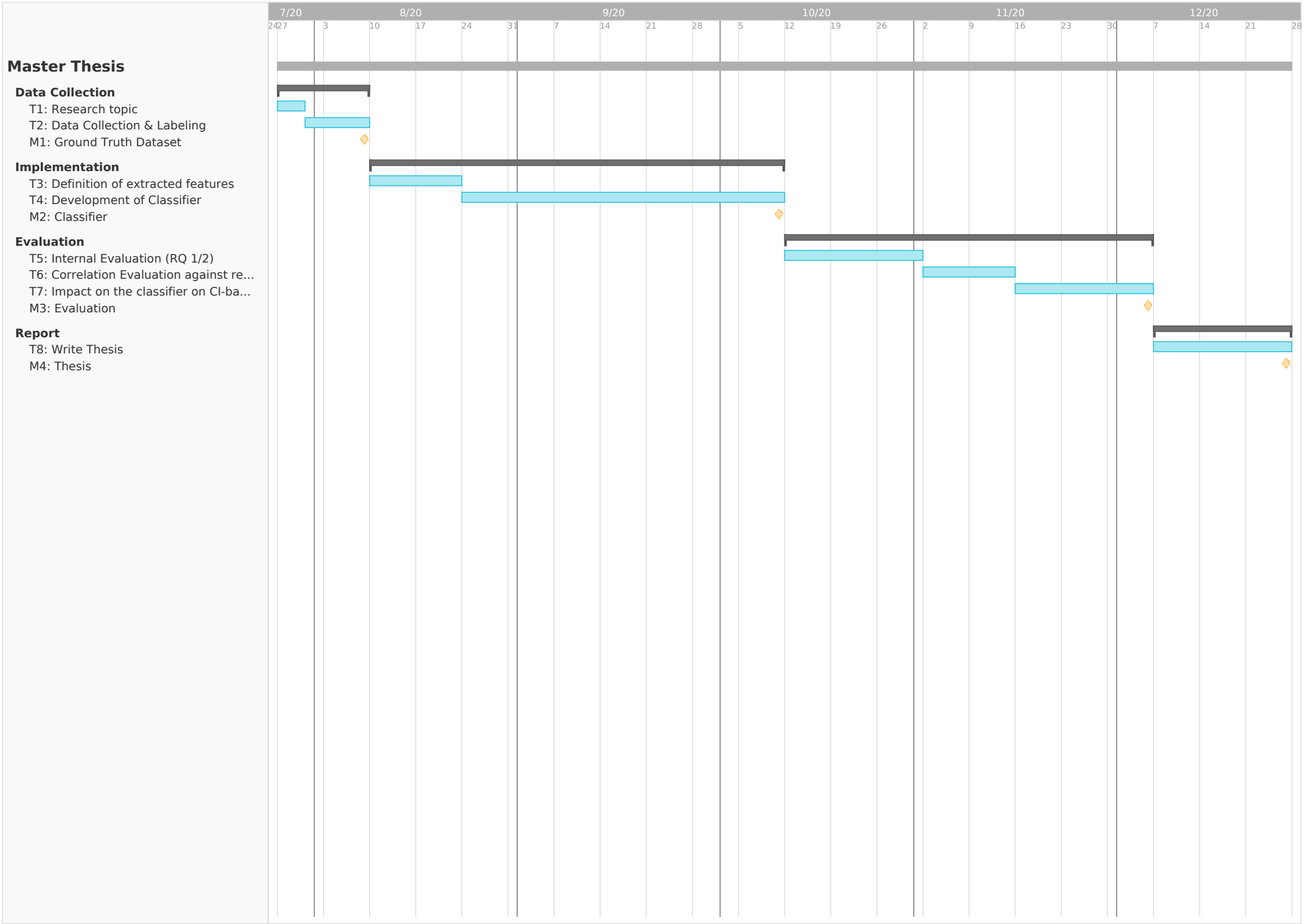
TODO ▷*We need to clarify what to write here... I am not an assistant to Harald anymore :)*◁
**Professor**:
Prof. Dr. Harald C. Gall

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

**Signatures:**

Timothy Zemp

Prof. Dr. Harald C. Gall


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
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

## 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.