# Automatic Means of Detecting Warning Signs in Software Evolution

Martijn Endenburg

April 10, 2014

Part-time programme
Master's Software Engineering

Supervisor
*Rutger Pannekoek*

Organization
*VNU Vacature Media*
*De Persgroep Nederland*

Tutor
*Magiel Bruntink*

Publication status
*Confidential*



University of Amsterdam

# Revision history

| Version | Recipients |
|---|---|
| 20140324 | Rutger Pannekoek |
| 20140410 | Magiel Bruntink, Rutger Pannekoek |

# Contents

# Abstract

Lorem ipsum dolor sit amet.

**Keywords:** keywords, here

# Preface

# Chapter 1

# Introduction

## 1.1 Problem analysis

In closed source projects, many evolutionary events are dictated by the management. Events such as team size changes, changes in activity and progress are often the result of a change in, for instance, the hiring policy of the organization, or an approaching deadline. These events can be used to identify project scale and progress, which allows comparing and co-analyzing different projects.

In an open-source software (OSS) project, evolutionary events are mostly theoretical with little empirical validation. Changes in team size, activity, and the like are more organic. This means that different OSS projects cannot be compared in the same scale.

OSS projects are being used by many organizations in an industrial environment. This makes evaluating the survivability of OSS projects important. At VNU Vacature Media, developers extensively use OSS projects in their day-to-day work. The projects vary from developers tools, and utilities, to operating systems and full system stacks. These OSS projects include, but are not limited to, Ansible, Apache web server, MySQL database, and Ubuntu/Linux. Amongst the previously listed well-known projects, less known projects are used.

Automatically detecting warning signs in the evolution of OSS projects is useful if the people at the company uses OSS projects to support the business. This business can be anything that is valuable to a company. The method of automatically detecting warning signs can aid in selecting OSS projects for new purposes. For instance, selecting a healthy project that has a bigger chance of surviving is more suitable because it is able to support the business for a longer period than a project of which the activity is decreasing. Additionally, the method can be used to monitor projects that the people at the company are using. In case serious anomalies are detected, a decision can be made to look for alternatives and eventually replace projects that have a lower chance of survival.

This study is a replication and extension of the research by Siim Karus *Automatic Means of Identifying Evolutionary Events in Software Development* [7]. It is extended by using a larger data set and attempts to find warning signs in the software evolution of OSS projects.

## 1.2 Research questions

The main research question of the thesis is (RQ1):

> *Can we use wavelet analysis to find objective warning signs in software projects leading to the end of code evolution?*

Wavelet analysis is the analysis of waveforms. It is an analysis tool that started in seismology. Its use is expanded to many other fields, such as, economics, geographics, audio and video processing, and others, to perform all kinds of digital signal processing (DSP).

Software evolution comprises various measures, such as lines of code, number of developers, etc. These metrics can be put in the frequency domain of a waveform, whereas the age in months of a project can be put in the time domain of a waveform. This way, wavelet analysis can be used to analyze software evolution.

As wavelet analysis is a tool that is relatively easy to automate, it provides automatic detection of patterns in software evolution.

In order to find the answer to the main question, the following questions will be answered:

*What patterns can be found using wavelet analysis?* (RQ2)

*Under what conditions does wavelet analysis succeed or fail in detecting evolutionary events?* (RQ3)

## 1.3 Outline

Chapter 2 will provide more information on prior research in the field of OSS survivability. In chapter 3 the method and approach of the research is discussed. Next, in chapter 4, the research execution itself is described. Chapter 5 presents the results of the research. Chapter 6 discusses the results after analysis and presents the conclusions of the research.

# Chapter 2

# Background

The survivability of OSS projects has been of interest by many researchers. In an industrial environment, a project is considered successful if it is completed within time and budget constraints. In OSS projects, these constraints do not always exist. The indicators of success for OSS projects differ from the one study to the other.

In a study by Samoladas et al. a method for survival analysis on OSS projects was proposed. These methods were employed to predict the survivability of the projects by examining their duration, combined with other characterstics such as application domain and number of committers. Although these metrics give insight in the survivability chances of a project, it was also found that adding a developer to the team of contributors increased the survivability of the project substantially. This relation cannot be linear; there must be a point where adding a developer to the project has no (substantial) increase of survivability.

The authors of the paper proposed two main research issues to be addressed in the future. The first one is to add more projects to the study with possibly a different categorization. And second, the effects of more project parameters, such as programming language should be examined. This is not trivial since typically more than one language is used in each project [12].

A study by Raja and Tretter on defining a measure of OSS project survivability. They have been looking for vitality of OSS projects: the ability of a project to grow and maintain its structure in the presence of perturbations. They identified three dimensions of project viability: vigor – the ability of a project to grow –, resilience – the ability of a project to recover from disturbances –, and organization – the structure exhibited in the project. These dimensions represent three distinct characteristics of project viability [11]. This measure is of use to determine survivability of a project, however, it is a snapshot of a point in time. It does not take into account the events prior to this point in time. Nor does it enable prediction of survivability in the near future. Therefore, it can be challenging to get an objective view on a project's survivability as a representative point in time has to be selected.

Crowston et al. identified measures that can be applied to assess the success of OSS projects. The authors used the DeLone and McLean model of information systems success to evaluate OSS project success [6]. The aspects identified by DeLone and McLean are elaborated; output of systems development – it is believed that a project that has a high frequency of releases is healthy –, process of systems development – the number of developers, the individual level of activity, and cycle time (time between releases) –, and project effects – employment opportunities of the contributors, individual reputation, and knowledge creation. In this study it was found that many of these aspects are indicators of OSS project success [4]. Although the cycle time is an aspect that could be measured automatically, the other aspects are *soft*. Soft in the way that it is hard to get into numbers. This makes automatic evaluation and analysis on these aspects very hard.

Another study conducted by Crowston et al. extends the previous study by using Free/Libre Open-Source Software (FLOSS) projects. They found that the number of developers as a simple count of developers is a somewhat flawed number as it aggregates the number of developers leaving and the number of developers joining a project. A 'churn' of the developers or a 'tenure' of individuals would

be more appropriate. This study took projects from one source code repository, SourceForge, instead of different repositories [5].

A study conducted by Wang has shown that warning signs can be found in six crucial factors of OSS projects success: developer participation effort, developer service quality, software license restrictiveness, targeted users, community social network ties, and community quality of social ties [13]. These factors play a role in OSS project success, however, these factors are not easily analyzed automatically.

Karus explored a method known as *wavelet analysis* to analyze software evolution data. It is a method that is easily automated. The wavelet analysis interprets evolution data as a series of signals and is able to find sequences in this signal. The sequences of multiple projects can be compared in order to find recurring patterns. Karus was able to detect close to 1,000 similar patterns across 27 OSS projects. His conclusion is that wavelet analysis can be a powerful tool for identifying evolutionary events [7].

In this research, we aim to add semantics to Karus' findings in a sense to be able to tell whether a specific pattern or event can be interpreted as a warning sign. A warning sign is a pattern or event that has a negative effect on the evolution of an OSS project.

# Chapter 3

# Research method

## 3.1 Wavelet transform and analysis

The word *wavelet* literally means 'small wave'. It has its origins in the theoretical continuous wave; e.g., a wave that lasts from minus infinity to plus infinity. A wavelet is a sample of a continuous wave that starts and ends at some points in time (i.e., is finite).

A wavelet is a signal in a 2 dimensional space. Each dimension is a domain. The domains are the *time domain*, visualized at the $x$ axis, and *frequency domain*, visualized at the $y$ axis. In the case of an audio signal, the analogies are easily made. In other fields, any observable property can serve as either frequency or time value. As long as the relation between the two properties is functional within the context of the signal.

Wavelet transformation is the sampling of a signal at different intervals which gives a natural means of scaling the signal. Wavelet analysis is the analysis of signals by decomposing the signal into wavelet coefficients (also known as shift coefficients) and scaling coefficients (also known as filters).
The decomposition can be repeated on the scaling coefficients until the number of resulting wavelet coefficients is smaller than the filter length.

*Shifting* is the operation of moving the wavelet through the time domain. This operation is also known as *translating* the wavelet. *Filtering* is the operation of scaling the wavelet in the frequency domain. This operation is also known as *dilating* the wavelet.

The main advantages of wavelet transform in time series analysis are:

- Scaling coefficients allow fuzzy matching as differences in details are "smoothed out".
- Filter coefficients allow detection of small anomalies in series.
- Transform levels make series of different lengths or scale comparable.

### 3.1.1 Haar filter

In this study and in the study by Karus, the *Daubechies* filter of length 2 (also known as *Haar* wavelet), due to its simplicity and simple interpretation.
We apply discrete wavelet transform, meaning we use discrete shift when matching the series with the wavelet. The scaling coefficients of discrete Haar wavelet transform can be interpreted as a smoothed curve of the series. The wavelet coefficients show temporal variations.

Naively spoken, the Haar transform is a means of digitalizing an analogue signal. It still serves this purpose in many devices we use in a day-to-day basis. The Haar transform sampling can be performed multiple times. The number of times depends on the resolution of the analogue signal. The higher the analogue resolution, the more sampling transforms are possible.

It starts by computing the integral of the full time series signal. It then divides the signal into two equal parts over the time domain. For each part it computes the integral. The difference of the two levels is recorded. Thus, for each increase in detail (i.e., each re-division of the signal) the extra information that is added is captured in that level. This process can be repeated until there is insufficient time resolution to add more information. This way, by repeatedly dividing the signal into a more detailed signal, we can get arbitrarily close to a continuous signal.

The result after transformation is a multi-layered sequence of values, where each layer represents a decomposition level.

### 3.1.2 Wavelets in software evolution

The use of wavelet transformation in the analysis of software evolution removes the factor of project size and enables comparing projects equally to find similar sequences.

In software evolution, the signal can be any measurable property of an evolving entity, such as team size, lines of code, number of commits, etc. These properties can be measured at a given time series, such as age in months, or even a non-time-related series such as lines of code. Measuring these properties over the evolution of a project gives a series of signals. Software repositories are a source of signals in software evolution. This allows wavelet transform be applied directly for mining software repositories.

# Chapter 4

# Research

## 4.1 Data

The evolution data of 250 OSS projects was gathered from Ohloh.net[1]. Ohloh is a project by the initiative of Black Duck Software[2]. The Ohloh universe contains approximately 600,000 open-source software projects in January 2014. The evolution data of all these projects is tracked and monthly analyzed by Ohloh.net.

### 4.1.1 Data gathering

The data was gathered using the tool *OhlohAnalytics* by Magiel Bruntink [3]. The tool is created as part of the research "*An Initial Quality Analysis of the Ohloh Software Evolution Data*" by M. Bruntink. This tool provides us with an initial data set of more than 10,000 OSS projects gathered in July 2013.

The following steps are done in the data gathering process:

1. Data collection.
2. Validation and cleaning.

#### Data collection

During the data collection step the names of the projects tracked by Ohloh are gathered. Then for each project, monthly analysis facts (so called *factoids*) are gathered. These facts include: size facts (lines of code (LOC), blank lines, commented lines), activity facts (number of commits, number of contributors, LOC added/removed), enlistments (i.e., source code repositories for the project), and programming languages.

#### Validation and cleaning

The initial validation step does sanity checks and filtering on the collected data. It filters out the projects that have a wrongly configured repository, has missing data, negative LOC added, negative LOC removed, negative LOC, negative number of commits, negative number of contributors, or have inconsistent values where the difference between the previous month and current month numbers are inexplainable.

The resulting set is a set of valid factoids for the 10,000 projects.

### 4.1.2 Data selection

The data selection procedure consists of the following steps:

1. Project coverage computation.
2. Master data filtering.
3. Sample selection.

[1] http://www.ohloh.net
[2] http://www.blackducksoftware.com

#### Project coverage computation

The cleansed set of 10,000 projects is analyzed by computing the project's evolution coverage. For each project, the difference between the minimum and maximum age in months is taken against the number of months present in the data set. This way, we have the percentage of the coverage between the mininum and maximum age.

#### Master data filtering

The next step is to filter out the projects from the master data that do not have a 100% coverage or that have less than 12 data points (months). The result of this step is a filtered master data set that is the input of the next step.

#### Sample selection

A sample of 250 projects need to be selected from the 10,000 projects gathered by the OhlohAnalytics tool. This sample must be a representative selection of the projects tracked by Ohloh.net. For this selection process the tool *SampleSoftwareProjects* was used. The SampleSoftwareProjects tool was developed for the research by Nagappan et al. at Microsoft Research [10].

The tool takes the filtered master data set containing facts of interest about the projects tracked by Ohloh.net to characterize these projects. These facts include activity facts (user count, average rating, last updated), project age, size facts (LOC), and language facts. The projects are given a score by each of these dimensions. Taking one sample project as base line the script evaluates projects from the master set and adds them to the sample if it increases the overall coverage to be close to that of the master set as a whole. The resulting data set is aimed to be 100% representative.

For this study we took the project *Mozilla Firefox* as basis for the sample. The resulting data set of 250 projects has a representativeness of 100% of the Ohloh.net universe, according to the SampleSoftwareProjects tool.

## 4.2 Metrics

The following metrics are used and measured monthly:

**Age in months**
: This is the number of months since the first factoid of a project.

**Active developers**
: At Ohloh.net this is called the number of contributors and is the actual number of developers that have made at least one commit in the month of the factoid.

**Lines of code (LOC)**
: For this study we use the same definition of LOC as in the original study by Karus; blank lines and commented lines are also included in the LOC counts. The reason for this is that we want to be able to measure activity and a comment added or removed is also considered project activity related to code.

**Code Churn**
: Is the sum of LOC added, LOC removed, and LOC modified. The modified LOC is not tracked at Ohloh.net. Modified LOC is aggregated in LOC added and LOC removed; a modified LOC is basically a line that is both removed and added.

The variable we used in the time domain is 'Age in months'; in the frequency domain this is 'LOC', 'Code Churn', and 'Active Developers'. Table 4.1 shows which variable in the time domain is plotted against what variable in the frequency domain.

Table 4.1: Evolution signals

| Time domain | Frequency domain | Description |
|---|---|---|
| Age in months | LOC | LOC per month. |
| | Code Churn | Churn per month. |
| | Active Developers | Contributors per month. |

## 4.3 Analysis

The analysis is done in the following steps:

1. Wavelet transform.
2. Similar sequence identification.
3. Similar sequence grouping.

### 4.3.1 Wavelet transform

During the wavelet transform steps, each project is analyzed separately using each combination as specified in table 4.1.

The data is prepared by sorting it by the time domain, indexing, and aggregate data where appropriate. Then the discrete wavelet transform is applied using the Haar filter.

The resulting data structure is saved in a file representing the transform data for a specific project. A file per coefficient (scale and filter) is saved with aggregated data for all projects. These files contain all values per level found for all projects.

### 4.3.2 Similar sequence identification

The results of the wavelet transforms are analyzed and identified as sequences. A sequence is a series of values with a minimum length of 3 values and a maximum length of 65. The sequences of each project are compared to the sequences of every other project. In case a similar sequence is found, it is recorded. Similar is having the same series of values with a maximum deviation of 0.005.

All found similar sequences are saved in a file per coefficient.

### 4.3.3 Similar sequence grouping

The similar sequence grouping step takes the similar sequences found in the previous step and classifies the sequences by number of projects, number of occurances, lengths, etc. The resulting file enables selection of sequences by these properties. The resulting data is used for partial manual, and partial automatic validation.

## 4.4 Warning signs

### 4.4.1 Dead projects

The validation of the sequence identification was done by first classifying a subset of the data set containing only dead projects. A dead project is defined as a project that had no change in LOC in the past 12 months.

The list of dead projects is then used to filter the sequences detected by these dead projects. The resulting set of sequences contains just the sequences in dead projects. Each sequence is related to other sequences in other projects which might be dead or not. These related projects are evaluated by checking if they exist in the dead projects list.

### 4.4.2 Dying projects

The related projects that are not 'dead' are extracted for analysis. Their related sequences will be analyzed. If a project from the 'not-dead' list has a sequence related to that of a 'dead' project, and the related sequence occurs at the end of the non-dead project's evolution data, then that project is a good candidate to be classified as a 'dying' project.

The resulting data set contains a list of projects that might be 'dying', the transform level at which the sequence was detected, and the number of times the sequence was matched against another sequence. This set is of use when manually validating the projects.

## 4.5 Validation

To validate whether the 'dying' projects are indeed dying, a manual investigation has to be made. For the resulting projects, the Ohloh.net project page can be consulted. The data set used contains data until July 2013, therefore, it is easy to confirm if the trend of 'dying' has continued up to April 2014. The activity report at Ohloh.net is very useful. It tells the amounts of commits and contributors in the past 30 days and in the past 12 months. Changes that show a decrease of over 30% in either commits, contributors, or both are in general a bad sign.

# Chapter 5

# Results

# Chapter 6

# Analysis and Conclusions

# Bibliography


[1] S. Andoutsellis-Theotokis, D. Spinellis, M. Kechagia, G. Gousios, Open Source Software: A Survey from 10,000 Feet, *Technology, Informations and Operations Management, vol. 4, Nos. 3-4, 2010, p187-347*

[2] M. Bruntink, Towards Base Rates in Software Analytics: Early Results and Challenges from Studying Ohloh, *Science of Computer Programming, Oct. 2013*

[3] M. Bruntink, OhlohAnalytics data set and analysis tools, *http://github.com/MagielBruntink/OhlohAnalytics, University of Amsterdam, 2013*

[4] K. Crowston, H. Annabi, and J. Howison, Defining Open Source Software Success, *School of Information Studies, 24th International Conference on Information Systems, 2003*

[5] K. Crowston, J. Howison, and H. Annabi, Success in FLOSS Development: Theory and Measures, *Software Process Improvement and Practice, 2006, no. 11, p123-148*

[6] W.H. Delone and E.R. McLean, Information Systems Success: The Quest for the Dependent Variable, *Information Systems Research, no. 3, 1992, p60-95*

[7] S. Karus, Automatic Means of Identifying Evolutionary Events in Software Development, *1063-6773/13 IEEE, 2013*

[8] S. Karus and M. Dumas, Code Churn Estimation Using Organisational and Code Metrics: An Experimental Comparison, *Institute of Computer Science Journal vol. 54, issue 2, 2012*

[9] S.G. Elbaum and J.C. Munson, Code Churn: A Measure for Estimating the Impact of Code Change, *IEEE Software Maintenance, 1998*

[10] M. Nagappan, T. Zimmermann, and C. Bird, Diversity in Software Engineering Research, *ESEC/FSE'13, August 1826, 2013, Saint Petersburg, Russia, ACM 978-1-4503-2237-9*

[11] U. Raja and M.J. Tretter, Defining and Evaluating a Measure of Open Source Project Survivability, *0098-5589/12 IEEE Transactions on Software Engineering vol.38 no.1, jan-feb 2012*

[12] I. Samoladas, L. Angelis, I. Stamelos, Survival Analysis on the Duration of Open Source Projects, *Information and Software Technology 52, 2012, p.902-922*

[13] J. Wang, Survival Factors for Free Open Source Software Projects: A Multi-stage Perspective, *European Management Journal 2012, no. 30, p352-371*