Automatic Means of Detecting Warning Signs in Software Evolution

Martijn Endenburg

March 24, 2014

Part-time programme
Master's Software Engineering

Supervisor
Rutger Pannekoek

Organization
VNU Vacature Media
De Persgroep Nederland

Tutor
Magiel Bruntink

Publication status
Confidential

-DRAFT-

University of Amsterdam

Revision history

Version	Recipients
20140324	R. Pannekoek

-DRAFT-

Contents

Abstract **3**

Preface **4**

1 Introduction **5**
1.1 Problem analysis 5
1.2 Research questions 5
1.3 Outline 6

2 Background **7**

3 Research method **9**
3.1 Wavelet analysis 9
3.2 Data 9

4 Research **10**
4.1 Data 10
4.1.1 Data gathering 10
4.1.2 Data selection 10
4.1.3 Data validation 11
4.2 Metrics 11
4.3 Analysis 11
4.3.1 Data aggregation 12
4.3.2 Wavelet transform 12
4.3.3 Similar sequence identification 12
4.3.4 Similar sequence grouping 12

5 Results **13**

6 Analysis and Conclusions **14**

Bibliography **15**

-DRAFT-

Abstract

Done and good.

Keywords: keywords, here

-DRAFT-

Preface

-DRAFT-

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.

The evolution of an open-source software (OSS) project is more organic. Events that shape the OSS project's evolution are mostly theoretical with little empirical validation. This means that different projects cannot be compared in the same scale.

OSS projects are being used by many organizations in an industrial environment. The number of organizations that use OSS projects is increasing. This makes evaluating the survivability of OSS projects more 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 server and client, and Ubuntu Linux.

Amongst the previously listed well-known projects, less known projects are used. Therefore, it is valuable to be able to determine or even predict the survivability of these projects. The detection of warning signs in and the monitoring of their evolution is useful to decide whether to choose for or continue to use an OSS project.

This study is a replication of the research described in the paper *Automatic Means of Identifying Evolutionary Events in Software Development* by Siim Karus. In this study, the evolution data of 250 OSS projects is analyzed to find patterns that can be interpreted as warning signs.

1.2 Research questions

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

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

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

RQ2
What patterns can be found using wavelet analysis?

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

DRAFT

1.3 Outline

Chapter 2 will provide more background and context about the research subject. 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.

-DRAFT-

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. 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. In addition, 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 [11].

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 [10]. Although this study is of use to determine survivability of a project it is a snapshot of a point in time, whereas the analysis of time series may provide a more objective judgement of project survivability.

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. 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. It was found that many of these aspects are indicators of OSS project success [4].

Another study conducted by Crowston et al. extended 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. Furthermore, this study took projects from one source code repository, SourceForge, instead of different repositories [5].

A study conducted by Wang has shown that early warning signs can be found in six crucial factors of OSS projects success: developer participation effort, developer service quality, software license restrictive-

DRAFT

ness, targeted users, community social network ties, and community quality of social ties [12]. However, warning signs in these factors are not easily detected automatically.

Taking into account the measures and metrics used in above studies, combined with the measures and metrics used by the author of the original research paper, we can see what data is needed to replicate the research by Karus [6].

-DRAFT-

Chapter 3

Research method

3.1 Wavelet analysis

The research questions mentioned in section 1.2 are based on the observations that wavelet analysis is used in other disciplines to analyze time series. For example, wavelet analysis has been a successful method in economics and (digital) signal processing (DSP).

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 (time series) 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.

In replicating the research by Karus we use the same Debauchies 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.

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

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

In the case of 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 days, 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 or time series in software evolution. This allows wavelet transform be applied directly for mining software repositories.

3.2 Data

The original research was conducted by using a data set of 27 OSS projects. In this study a data set of 250 OSS projects is used. The use of a larger data set will in theory have a better spread of all OSS projects. Additionally, the results of the research with a larger data set can be interpreted more generally and with fewer noise.

DRAFT

Chapter 4

Research

4.1 Data

Data was gathered from Ohloh.net[1]. Ohloh is a project by the initiative of Black Duck Software. 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 aggregated.

4.1.1 Data gathering

For the data gathering and validation we used the tool *OhlohAnalytics* created by Magiel Bruntink [3]. This tool was created for the replication 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 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), language (main programming language, other 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.

4.1.2 Data selection

From the 10,000 projects gathered by the OhlohAnalytics tool, a sample of 250 projects have been selected. This sample must be a representative selection of the projects tracked by Ohloh. For this selection process a script called *SampleSoftwareProjects* was used and provided with the OhlohAnalytics tool. The SampleSoftwareProjects tool was developed by Nagappan et al at Microsoft Research [9].

The script takes a master data set containing interesting facts about the projects tracked by Ohloh. These facts include activity facts (user count, average rating, last updated), project age, size facts (LOC), and language facts. The projects are scored by each of these dimensions.

[1] http://www.ohloh.net

DRAFT

Table 4.1: Time series and signals

Time series	Signal	Description
Age in days	Active developers	Team size over time.
	Lines of code	LOC size over time.
	Code churn	LOC change over time.
Active developers	Code churn	LOC change per team size

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% according to the SampleSoftwareProjects script.

4.1.3 Data validation

As a final step we have validated the 250 projects for data completeness. This is not a trivial step as we want to be able to analyze the project's evolution data. Therefore, the more complete this data is, the more likely we will be able to detect patterns. For all 250 projects we counted the number of monthly factoids before and after cleansing and computed the percentage of evolution data coverage per project. For the least covered project we have 5.1% of its evolution data, the highest coverage is 99.7%. The average coverage is 78.6%. 19 projects are covered for 33% or less. 156 projects are covered for more than 80%.

4.2 Metrics

The following metrics are used and measured monthly:

Age in days
: This is the number of days since the first factoid of a project. The factoids are monthly statistics, thus the age in days is determined by the date of the factoids. We have taken a constant of 22 working days per month. The average month has 30.4 days, divided by 7 gives 4.34 weeks, multiplied by 5 working days gives 21.7 working days per month.

Active developers
: At Ohloh 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 but is aggregated in LOC added and LOC removed. A modified LOC is basically a line that is both removed and added.

For time series dimensions we used *Age in days* and *Active developers*. For signals we used *Active developers*, *LOC*, and *Code churn*. Table 4.1 shows what time series are plotted against what signals.

4.3 Analysis

The analysis of the data is done per project, per time series, per signal series. For 250 projects and 4 two-dimensional series this means that were a total of 1,000 analysis runs.
The analysis was conducted in the following steps:

DRAFT

1. Data aggregation.
2. Wavelet transform.
3. Similar sequence identification.
4. Similar sequence grouping.

4.3.1 Data aggregation

The first step of the analysis is aggregregation of the data by the time series under analysis. For instance, if there exists multiple factoids for the same time series value, this data has to be aggregated. Depending on the signal variable the aggregation was a summation, mean, max, or min value. The output of the aggregation step a set of data for one project consisting of one data point per time value.

4.3.2 Wavelet transform

The aggregated data is fed to the discrete wavelet transformation. This step is divided into two ways of wavelet transformation each using a different coefficient. One wavelet transform is done using the filter coefficent, the second using the scaling coefficient. The results of the wavelet transform step are sequences of project evolution data which are patterns of interest. These sequences have characteristics such as length and amplitude that are beyond a predefined threshold.

4.3.3 Similar sequence identification

The results of the wavelet transforms for each dimension are analyzed and compared to find similar sequences in different projects.

4.3.4 Similar sequence grouping

-DRAFT-

Chapter 5

Results

-DRAFT-

Chapter 6

Analysis and Conclusions

-DRAFT-

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

[7] 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*

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

[9] 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*

[10] 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*

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

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

-DRAFT-