Universiteit Utrecht

MBI Master Thesis

To Fail or Not to Fail: A Z-score for Open Source Projects

Author:	Shaheen Syed
E-mail:	s.a.s.syed@uu.nl
Student number:	3789853
First Supervision:	dr. Slinger Jansen
	slinger.jansen@uu.nl
Second Supervision:	dr. ir. Peter de Waal
	p.r.dewaal@uu.nl

Abstract

This research is focused towards the prediction of Free Libre Open Source (FLOSS) project failure and non failure. By using the characteristics of open source projects, referred to as determinants, we strive to create a classification model that predicts project failure and additionally, project non-failure. Examples of project characteristics are the number of developers, the number of downloads, the number of releases and other indicators that relate to success or failure. In order to arrive at such classification model, we adopted the method to predict corporate bankruptcy by E. Altman, also known as the Z-score in economic context. The research method employed by Altman provides us with the necessary steps towards a classification method for FLOSS project failure. That is, creating a sample based on *a priori* groupings (failed and non-failed projects), possibly one year prior to the event, and performing multiple discriminant analysis to create a linear function that best discriminates between the chosen groups. This enables project administrators, or other stakeholders of open source projects, to assess their project outcome and possibly steer it into a more successful outcome. The Z-score model for open source projects, as we have named it, is able to predict 65% of the cases correctly. Meaning, any new project can be classified with 65% accuracy for its outcome, being a failure or non-failure. Furthermore, the model works best for predicting open source project failure, as approximately 70% of the cases can be correctly predicted. Although we believe that essential indicators for open source success or health are hard to measure in numeric values, making the classification model only a reflection of data that can be operationalized, we believe this is first step towards a concrete model to assess open source projects.

Contents

Abstract ii

Table of Contents iv

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Problem Statement . . . 2
- 1.2 Objective . . . 2
- 1.3 Relevance . . . 3
  - 1.3.1 Scientific Contribution . . . 3
  - 1.3.2 Societal Contribution . . . 3
- 1.4 Thesis Outline . . . 4

2 Research Method 5
- 2.1 Research Questions . . . 5
- 2.2 Research Design . . . 6
- 2.3 Literature Review . . . 10
- 2.4 Statistical Analysis . . . 11

3 Theoretical Framework 13
- 3.1 Altman's Z-score . . . 13
- 3.2 Success and Failure Determinants within Open Source (FLOSS) Projects . . . 15
  - 3.2.1 Success and Failure in Initiation and Growth Stage . . . 16
  - 3.2.2 Reclassification of Success and Failure in Initiation and Growth Stage . . . 18
  - 3.2.3 Survival Factors in Initiation and Growth Stage . . . 18
  - 3.2.4 FLOSS Success in the Context of Traditional Information System Development . . . 20
  - 3.2.5 FLOSS Success Measures and their Determinants . . . 23
  - 3.2.6 Survival Analysis on Future Development of FLOSS Projects 26
  - 3.2.7 FLOSS Project Activity Indicators and Classification . . . 26
- 3.3 Definition Failed FLOSS Project . . . 27
  - 3.3.1 The Definition . . . 28

4 Data Preparation **29**
4.1 Relevant Determinants 29
4.2 Availability of Data 29
4.2.1 Availability of Determinants 31
4.2.2 Discarded Determinants 33
4.3 Overview of Determinants 35
4.4 Extra Determinants 36

5 Data Extraction **37**
5.1 Sample Creation 37
5.2 Classification 41
5.3 Used Determinants 41

6 Analysis **44**
6.1 Descriptives 44
6.2 Multiple Discriminant Analysis 44
6.2.1 Assumptions 45
6.2.2 Stepwise Analysis 46
6.3 Results 46
6.3.1 Box's M 48
6.3.2 Eigenvalues 48
6.3.3 Standardized Canonical Coefficients 49
6.3.4 Canonical Discriminant Function Coefficients 49
6.3.5 Classification Results 50
6.3.6 Histograms 52
6.3.7 Cut-off Point 52

7 Conclusion & Discussion **54**

References **57**

Appendix **59**

A Statistical output **60**

List of Figures

2.1 Process Deliverable Diagram 8

3.1 FLOSS lifecycle stages (Wang, 2012) 28

5.1 Schematic view of the data collection phase using SRDA 38
5.2 Schematic view of the data collection phase using SourceForge APIs 39
5.3 Schematic view of the data collection phase extracting data from SourceForge project pages 41
5.4 Sample creation method 42
5.5 Classification method to label projects within our sample as 0 = failed and 1 = non-failed in subsequent year 42

6.1 Histograms showing the distribution of discriminant scores for failed and non-failed FLOSS projects 52
6.2 Linear distribution of FLOSS Z-score with predictors (a) *number of programming languages* and (b)*versioning control write transaction* 53

List of Tables

2.1 Activity table of research steps 9
2.2 Concept Table 10

3.1 Description of variables used by Altman's Z-score 14
3.2 Z-score classification and prediction accuracy of distressed firms with cutoff point 2.67 (1.81 cutoff in parenthesis) 15
3.3 Six FLOSS success/failure classes and their methods of operationalization (English & Schweik, 2007). 17
3.4 Overview of survival measures for FLOSS projects categorized into three classes 20
3.5 Summary of concepts for Information System success in FLOSS context (Crowston et al., 2006). 22
3.6 Overview of Midha and Palvia's (2012) results to determine success factors for OSS projects. 24

4.1 Overview of determinants that may affect FLOSS project outcomes 30
4.2 Discarded determinants 34
4.3 Determinants selected for further analysis 35

6.1 Tests of equality of group means 47
6.2 Box's M test results 48
6.3 Eigenvalues 49
6.4 Standardized Canonical Discriminant Function Coefficients . . . 49
6.5 Canonical Discriminant Function Coefficients 50
6.6 Classification results 51
6.7 Classification results with equal prior probabilities 52

A.1 Descriptives of determinants for total sample 61
A.2 Descriptives of determinants for failed group 62
A.3 Descriptives of determinants for non-failed group 63
A.4 Pooled within groups matrix 64

Chapter 1

Introduction

Free/Libre Open Source Software (FLOSS) covers a diversity of software and development approaches. In general, Free/Libre Software or Open Source Software refers to software released under a license that permits the inspection, use, modification and redistribution of the software's source code (Crowston, Wei, Howison, & Wiggins, 2012). There are, however, distinctions between The Free Software movement and the Open Source movement that set both development communities apart. Richard Stallman, founder of the GNU project[1], recapitalizes these as follows (Stallman, 2010):

" The fundamental difference between the two movements is in their values, their ways of looking at the world. For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, Open source is a development methodology; free software is a social movement. For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution. "

FLOSS projects differ in many ways compared to the principles and practices advocated by traditional software engineering (SE) (Feller, Fitgerald, Hissam, & Lakhani, 2005). Software development is done globally and performed by members of their respective open source community, who are culturally and geographically dispersed. G. Lee and Cole (2003) characterize this as community-based development. Besides that, developers often participate without monetary rewards and little or no intrinsic management. In addition, despite their perceived lack of organizational support, there are FLOSS projects with large numbers of developers working collaboratively to create software of complexity and quality that rivals their commercial counterparts (Raymond, 1999; Kuwabara, 2000). Forges such as SourceForge.net[2] and Github.com are often used to organize FLOSS development efforts.

As a consequence, large numbers of FLOSS projects are now being used by thousands, and even by millions of end-users, ranging from web-servers (e.g.,

[1] http://www.gnu.org

[2] SourceForge.net is the world's largest Open Source software development web site, with the largest repository of Open Source code and applications available on the Internet.

Apache), e-mail servers (e.g., Sendmail), programming languages (e.g., Ruby, Perl, Java, Python, PHP), and operating systems (e.g., Linux, Unix). Some of these FLOSS projects, e.g. Linux, Mozilla web browser and Eclipse entail millions of lines of code and thousand of active developers. With this growth, there has been a concurrent increase in research examining the phenomenon.

1.1 Problem Statement

In recent years, the traditional paradigm of software innovation based on intellectual property rights has been challenged by the emergence of FLOSS (Comino, Manenti, & Parisi, 2007). Various studies have been devoted to understanding FLOSS' underlying structure and to assessing the potential benefits of a more widespread adoption of FLOSS, possibly fueled by the popularity of Mozilla, Apache, Linux etc. Apart from these successful FLOSS projects, there is a vast amount of software that is not publicly known or has started but ended rapidly. Various studies have shown that successful FLOSS projects attract more talented developers, more users and even sponsors (Hann, Roberts, & Slaughter, 2004; Lerner & Tirole, 2002). As a consequence, the need to assess FLOSS projects can help all project stakeholders, e.g. project administrators, sponsors, developers, users etc., to get insights in the health of their project and are able to act upon accordingly. To conclude, the problem statement addressed in this research can be summarized as:

Due to the strong emergence of Open Source Software in commercial companies, heavily depending on the survival of their software systems, the need to assess Open Source software is of high value to its stakeholders. Furthermore, Open Source project administrators need to have insights in their software in term of being successful or not, easily assessing their software can aid in this respect and possibly steer it into a better project. Current research focusses solely on certain success determinants, such as the number of downloads, they however fail to operationalize success factors into a simple formula to assess open source projects. Creating a formula, based on the concept of the z-score, extends current research in this matter.

1.2 Objective

This research's objective is to define a measurement scale that assesses the health of FLOSS projects by taking into account relevant success and failure characteristics. It uses the concept of a Z-score, a formula in economics to predict whether a company will file for bankruptcy or not (Altman, 1968), and translates variables (e.g. characteristics of a company or project) into a scalar value by using discriminant analysis. The original Z-score's objective is to predict corporate bankruptcy, similarly, the objective of a Z-score for FLOSS projects is to predict FLOSS project failure prior to the event happing.

The FLOSS Z-score will produce a scalar value where a cutoff point (point that divides the chosen groups) classifies a project as either becoming a failure or not. Additionally, projects that are not in the distressed zone, being above the

cutoff point, can be classified to prolongate its existence, and more interestingly, projects that score much higher than the cutoff point, being in the upper region of the non-distressed zone, can be seen as more successful or healthy projects. Despite these additional uses, the main objective and its intended purpose is to predict FLOSS project failure, preferably one year prior to the event happing.

1.3 Relevance

This section discusses the relevance of the problems investigated in this work and the research contribution. Two perspectives are employed, a scientific perspective that outlines the benefits to the academic community, and a societal perspective that describes the worth of this research to society in general.

1.3.1 Scientific Contribution

From a scientific point of view, this research contributes to the body of empirical knowledge on the topics of FLOSS success, FLOSS failure, and to some extent, FLOSS health. In addition, it provides a valuable instrument for measuring FLOSS project outcome, and more specifically, FLOSS project failure by using retrospective data. Several researchers have already defined success and failure determinants for FLOSS, e.g. (Crowston, Howison, & Annabi, 2006; English & Schweik, 2007; S.-Y. Lee, Kim, & Gupta, 2009; Midha & Palvia, 2012; Samoladas, Angelis, & Stamelos, 2010). This research will, however, extend their work by incorporating the determinants under study and translate them into a simple formula, like the Z-score in economic context (Altman, 1968), with respect to FLOSS project failure. In addition, researchers are often interested in identifying successes or failures in order to investigate the potential causes of success or failure. This research can also provide lessons and directions for future empirical research on FLOSS. Finally, assessing FLOSS survivability would be beneficial for educational purposes as universities incorporate participation in open source projects as part of their software engineering curriculum. Instructors supervising such courses are highly interested in their students participating in successful projects, rather than projects that are destined to fail (Stamelos, 2009).

1.3.2 Societal Contribution

This research contributes to society in general, and more specifically to FLOSS stakeholders, in numerous aspects which are listed below:

I Better management and more accurate identification of FLOSS project outcomes for project administrators. Ultimately, this could lead to a project's success both in terms of market penetration and technical achievements over time (Midha & Palvia, 2012).

II Better understanding of the chances of survivability for project administrators

III In cases where FLOSS projects are sponsored by third parties, it can be useful for sponsors to understand the return on their investment.

IV Having a practical measure for assessing the health of FLOSS projects is beneficial for companies who are willing to invest resources in new and relatively unknown projects.

V Volunteer programmers might be more interested in entering a project that has high chance to evolve than to fail and be abandoned. Being able to assess FLOSS projects can assist them in their choice.

1.4 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 will elaborate on the Research Method. A detailed description is provided on the various steps undertaken that led to the FLOSS Z-score model. Chapter 3 serves as a Theoretical Framework underpinning the determinants that will ultimately affect FLOSS project outcomes. We will discuss the current body of knowledge on FLOSS project success and failure, and try to distill operationalizations they have used. These in turn will serve as a foundation for our development of a FLOSS project Z-score. Additionally, we will describe the original Z-score that was developed for economic purposes by Altman and explain how it was developed. Also, we will define FLOSS project failure and non-failure. Chapter 4 describes the data preparation phase of our research method. We will elaborate on the determinants that were taken into account for further analysis, and why we discarded some of them. Chapter 5 discusses the data extraction phase. We will focus on the creation of our sample and how we collected data from various sources. We will provide details how we queried the archives, mined APIs and spidered project web pages. Furthermore, we describe how we classified projects that serve as a basis for performing multiple discriminant analysis. In Chapter 6 we will address the results of our research. More specifically, we will provide a step by step view of our statistical analysis and we will present our classification model with classification results. In Chapter 7 we will elaborate on the discussion part, conclude our research, and provide directions for improvements and future research.

Chapter 2

Research Method

This chapter will discuss our research question and the sub-questions involved that serve as a basis for this thesis. In addition, we will provide a detailed description of the various steps that try to answer our research questions.

2.1 Research Questions

The main research question formulated below addresses the need for FLOSS project stakeholders to assess their project in a simple manner. The use of available metrics that characterize their project will serve as input for the assessment. The assessment is based on the Z-score principles developed by Altman (1968). More specifically, the assessment is focused towards predicting FLOSS project outcome in a negative sense, i.e. the prediction of a failed FLOSS project. The main research question is therefore formulated as follows:

RQ **How can a model be created to predict FLOSS project failure by taking into account the project's own characteristics?**

The model, in this case a classifier, will be constructed by employing Multiple Discriminant Analysis (MDA). The classifier will serve as a means to classify FLOSS projects into either becoming failed or non-failed projects. In order to construct such a model, several sub-question need to be answered first, which are listed below:

S-RQ 1 **What determinants (variables) are important that affect FLOSS project outcomes, albeit in success or failure, and how are they operationalized?**

A FLOSS project has several characteristics that can positively or negatively affect its outcome. For example, number of developers, number of users, targeted audience etc. These characteristics are labeled as determinants, as they may or may not influence the project's outcome in the long run. It is, however, redundant to incorporate all project characteristics as some may even not influence the project. Answering sub-question 1 (S-RQ 1) will serve as the first step towards developing the Z-score model.

S-RQ 2 **How do we define and operationalize FLOSS project failure?**

In order to classify FLOSS projects into failed and non-failed, we need a concise and workable definition of FLOSS project failure and non-failure. What determinants can be used to label projects as failed, and more importantly, what values of these determinants are used to label them as such? For example, a determining variable would be the number of downloads, we operationalize this as projects that constitutes less than x downloads within a time-period of y months. The answer of sub-question 2 (S-RQ 2) will help to compose a sample for further statistical analysis. All projects that do not adhere to the definition of a failed project are labeled as a non-failed project.

S-RQ 3 **How do we deal with incomplete, inconsistent and poor quality of data?**

To construct a sample we need to collect data from various sources. We want to use a large selection of determinants and therefore need data on various characteristics of FLOSS projects. As we want to construct our sample as complete as possible, we need to obtain data on as much characteristics as possible and doing this for each project. The main problem lies within the fact that not all data is readily available. The data that can be collected need also be complete and reliable in order to construct a sample that represents the actual state of a project. We therefore need to check the completeness, consistency and quality of the data before proceeding. How to deal with cases where one or several of these conditions are not met is crucial for the validity of our proposed model.

2.2 Research Design

The development of the Z-score model for FLOSS projects will take several concurrent steps that will be discussed in this section. To provide an overview of these steps, a Process Deliverable Diagram (PDD) is constructed. A Process deliverable diagram (Weerd v.d. & Brinkkemper, 2008) is especially helpful when providing a clear overview of the various activities and its corresponding deliverables. The left-hand side of the diagram are the research activities which are based on the UML activity diagram (OMG, 2003). The right-hand side shows the corresponding deliverables that are based on the UML class diagram (OMG, 2004). Figure 2.1 depicts the PDD.

First, an extensive literature study is performed on the topics of FLOSS health, FLOSS success, FLOSS failures and the determinants that affect its outcomes. This literature will serve as a basis for the remainder of this research and is captured in a theoretical framework. Second, based on the theoretical framework, several determinants are important that can possibly affect FLOSS project outcomes. These determinants will be extracted and assessed on their availability based on (retrospective) SourceForge.net data. Another important step is to define project failure. As our main goal is to predict FLOSS project failure, a concise and working definition will be extremely important as the remainder of our research is based on it. Third, we will make use of SourceForge.net data and of its research database. Several scripts and queries will be constructed that are necessary for data extraction. The extracted data will be

cleaned and processed for further analysis. In addition, a sample will be constructed that forms the basis for statistical analysis. Finally, from the available data and by performing statistical analysis, mostly in perspective of Multiple Discriminant Analysis, a model (classifier) is constructed. This classifier, defined as a function, is able to predict whether a FLOSS project will fail or not, and can to some extent, indicate if a FLOSS project is healthy or not. The next step in this process is to validate the model on its accuracy, i.e. test to what extent the model is able to predict FLOSS project failure. This is done by performing cross validation techniques. For example, the jack-knife method for discriminant analysis can be used. One observation is excluded and a discriminant function is estimated. The latter is then used to classify the omitted observation; this observation may be classified correctly or incorrectly. Subsequently another observation is omitted from the first population and replaced by the observation which was excluded originally. Again the discriminant function is estimated and used to classify the omitted observation. This process is repeated for the entire sample. The relative proportion of misclassifications will then approximate the unbiased estimator of the Type I error.

To further explain the process visualized in Figure 2.1, Table 2.1 provides a succinct description of the activities involved in the research process, while Table 2.2 displays the definitions of the corresponding deliverables and concepts.

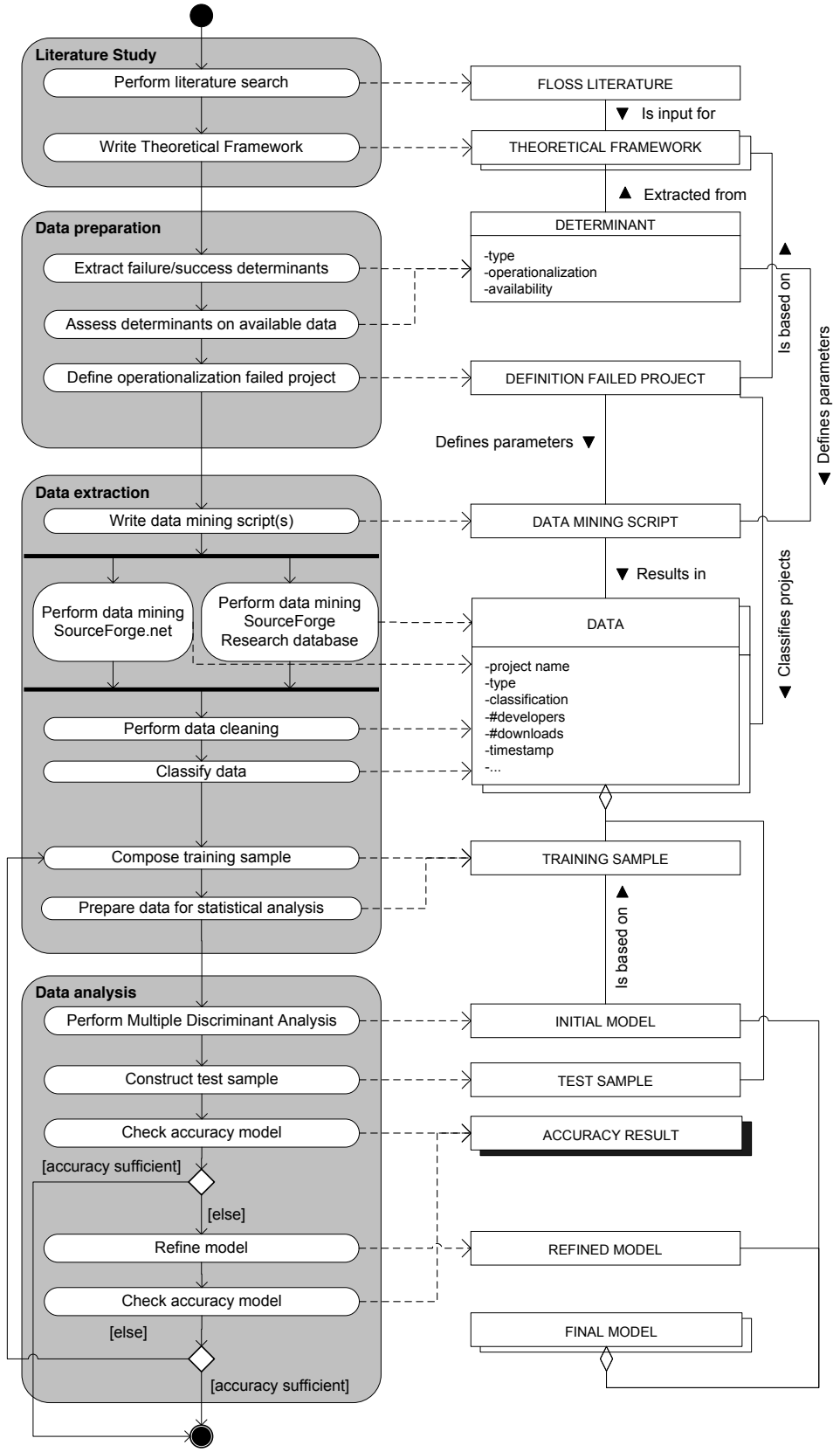


Figure 2.1: Process Deliverable Diagram

Activity	Sub-Activity	Description
Literature study	Perform literature search	Search for scientific LITERATURE on the topics of FLOSS success, failure, health, survival, impact etc.
	Write theoretical framework	Capture relevant literature and summarize in a THEORETICAL FRAMEWORK.
Data preparation	Extract failure/success determinants	Extract from THEORETICAL FRAMEWORK all relevant DETERMINANTs that can possibly affect FLOSS project outcomes.
	Assess determinants on available data	Check whether the DETERMINANT can be captured in either categorical or numerical value from SourceForge.net.
	Define operationalization fail project	Define a concise and workable DEFINITION of a FAILED PROJECT that forms the basis of further classification.
Data extraction	Write data mining script(s)	Write DATA MINING SCRIPT(s) that are executable in order to retrieve data from Sourceforge.net portal or research data base.
	Perform data mining SourceForge.net	Execute DATA MINING SCRIPT in order to retrieve DATA from SourceForge.net portal.
	Perform data mining SourceForge Research Data Base	Execute DATA MINING SCRIPT in order to retrieve DATA from SourceForge.net research database.
	Perform data cleaning	Check for abnormalities in the DATA and possibly use data cleaning methods.
	Classify data	Use the DEFINITION of a FAILED PROJECT to classify the projects in the DATA into failed and non-failed projects.
	Compose training sample	Compose TRAINING SAMPLE on a stratified random basis that form the basis for further statistical analysis.
	Prepare data for statistical analysis	Adjust or export TRAINING SAMPLE into workable data for appropriate statistical package.
Data analysis	Perform multiple discriminant analysis	Use the TRAINING SAMPLE as input and perform MDA to create a first INITIAL MODEL.
	Construct test sample	Compose TEST SAMPLE from the DATA on a stratified random basis to test the INITIAL MODEL's accuracy.
	Check accuracy model	Perform classification with the INITIAL MODEL on the TEST SAMPLE to validate the model's accuracy.
	Refine model	Refine the INITIAL MODEL's parameters to construct a new REFINED MODEL.
	Check accuracy model	Perform classification with the REFINED MODEL on the TEST SAMPLE to validate the model's accuracy.

Table 2.1: Activity table of research steps

Concept	Description
FLOSS LITERATURE	Academic literature on Free/Libre Open Source software that is focused on the topics of health, success, failure, survival and other related topics.
THEORETICAL FRAMEWORK	An overview of all related FLOSS LITERATURE that is summarized together and highlights results from previous studies.
DETERMINANT	A variable/factor/characteristic that may affect FLOSS project outcome.
DEFINITION FAILED PROJECT	A concrete definition when a FLOSS project can be labeled as failed.
DATA MINING SCRIPT	A programming script that can be executed to retrieve data stored in repositories.
DATA	The DATA on FLOSS projects with all values of the predefined DETERMINANTS.
TRAINING SAMPLE	A subset of the DATA that is used for initial statistical analysis.
INITIAL MODEL	A first or preliminary model that is able to predict FLOSS project failure.
TEST SAMPLE	A subset of the DATA that is used to test the INITIAL MODEL on its accuracy.
ACCURACY RESULT	The results of correct classification of the INITIAL MODEL on the TEST SAMPLE.
REFINED MODEL	An altered, and possibly improved version in terms of accuracy of the INITIAL MODEL
FINAL MODEL	The FINAL MODEL that best predict FLOSS project failure.

Table 2.2: Concept Table

2.3 Literature Review

In order to develop a comprehensive theoretical framework, a literature study was first conducted. The literature study follows the three-stage systematic process as defined by Levy and Ellis (2006). Their systematic literature review process is especially applicable to the challenges inherent in information systems research. More specifically, the information systems literature is comprised of diverse and interdisciplinary work (Webster & Watson, 2002), which may be a potential cause to overlook some important work conducted in other IS subdisciplines. The three-stage process as defined by Levy and Ellis comprises: Input, Processing and Output.

The literature study is based upon a concept-centric approach (Webster & Watson, 2002) and utilizes sources that are linked to keywords such as FLOSS health; FLOSS failure; FLOSS success; open source success; open source failure; FLOSS survival; open source determinants; FLOSS determinants and various combinations of these.

Two approaches were applied for obtaining relevant scientific literature that

were based on the work of Webster and Watson (2002). The first one was the usage of specialized search engines for scholarly work, and the second one was following the relevant references from articles already identified as pertinent. Two iterations of each approach were applied in the following order, a preliminary search engine search, a reference follow-up, another search engine search, and a final reference follow-up. The scholarly search engines used for the literature review were Google Scholar[1] and ISI Web of Knowledge[2].

In addition to the structured approach, a more explorative search was conducted on the aforementioned keywords by using the search engine of ScienceDirect[3] to obtain high quality journal papers from Elsevier. Furthermore, the search engine from Omega[4] was used to obtain additional papers that were not obtainable from scholarly sources only.

The next phase is to process relevant literature and to extract applicable knowledge from it. The output will provide a clear and logical structure of literature of the domain under study. This process was the second step in the work of Levy and Ellis (2006) and comprises: know the literature; comprehend the literature; apply; analyze; synthesize and evaluate. Ultimately, this forms the theoretical framework which Levy and Ellis (2006) label as output.

Approximately 50 scientific papers were found and analyzed upon their usefulness. Any relevant determinant that may affect FLOSS project outcome, positively and negatively, were extracted and summarized in our theoretical framework. As a result, the theoretical framework comprises determinants and describe how they affected FLOSS projects on the long run. Various definitions and operationalizations of success measures and project failure were documented as well.

2.4 Statistical Analysis

Besides a literature study, the ample part of this thesis is performed by applying statistical techniques on the gathered data. Several techniques exist for regressions and/or classification. Examples are linear regressions, logistic regression, classification trees, clustering, discriminant analysis and machine learning. Which one to use and which one provides us with best results depends on our intentions. We want an easy to use model that is able to predict a categorical variable. Furthermore, we want to take into account several variables and check which ones perform best by also taken into account the relationship between these variables. The same approach has been undertaken by Altman (Altman, 1968) when constructing the Z-score model for companies. The idea is to see what characteristics (variables) of companies best predict a so called categorical variable, which are distresses and non-distressed firms. It is similar to our approach, where we want to classify projects into failed and non-failed groups by using a set of variables and calculate which one best predict the groupings we defined upfront. The use of multiple discriminate analysis best suits our needs for the following reasons: MDA is a statistical technique used to classify an observation into one or several *a priori* groupings by utilizing the observation's

[1] http://scholar.google.com
[2] http://apps.isiknowledge.com
[3] http://www.sciencedirect.com.proxy.library.uu.nl/
[4] http://omega.library.uu.nl/seal/omegasearch.php

individual characteristics. It is primarily useful when the dependent variable appears in qualitative form, for example, distressed or non-distressed companies, successful or failed FLOSS projects. MDA in its most simple form attempts to derive a linear combination of characteristics which best discriminates between the groups. MDA has the advantage of considering all characteristics of the observation, as well as the interactions between them. Altman's Z-score uses two groups that reduces the complexity to just one dimension: resulting in a discriminant function in the form of $Z = v_1x_1 + v_2x_2 + \ + v_nx_n$. As we will see, the Z-score model is linear in that five measures are objectively weighted and summed up to arrive at an overall score. This score becomes the basis for classification into the *a priori* groupings, e.g. distressed and non-distressed firms.

Keeping in line with Altman's Z-score, as it most resembles our intentions, using its research method will suit our needs best. We therefore apply multiple discriminant analysis to construct the Z-score model for open source projects.

Chapter 3

Theoretical Framework

This chapter elaborates on the available literature that focuses on the success and failure determinants of FLOSS projects and how they are operationalized. We begin this chapter by discussing Altman's Z-score and how it was developed to predict corporate bankruptcy one year prior to the event. The various steps undertaken to create such measure will aid in understanding its origin. Subsequently, we will discuss various studies with respect to FLOSS project success and failure, and more specifically, the used variables (determinants) and their operationalization. These in turn will contribute to our initial list of variables to construct the Z-score for open source projects.

3.1 Altman's Z-score

Edward Altman developed the so-called Z-score model for assessing the distress of industrial corporations (Altman, 1968). The Z-score is a widely used model to predict corporate bankruptcy based on a number of variables. Altman's model originates from the work of Beaver (Beaver, 1967), who used ratio analysis for bankruptcy classification. Beaver's univariate analysis set the stage for Altman's multivariate analysis, who replaced ratio analysis by multiple discriminant analysis (MDA).

The development of the Z-score model originates from an initial sample of 66 corporations: 33 firms in each of the two groups. The distressed firms (group 1) filed for bankruptcy in the periods 1946 through 1965. The non-distressed firms (group 2) consist of a paired sample of manufacturing firms chosen on a stratified random basis by industry and size. This to cope with the not completely homogeneous group of distressed firms. Both groups had asset sizes ranging from $1 to $25 million. Firms in group 2 were still in existence at the time of the analysis. Also, the collected data are from the same years as those compiled for the bankrupt firms, being financial statements dated one annual reporting period prior to bankruptcy. A list of 22 potentially helpful variables, which must be seen as ratios of variables, was compiled for evaluation (see 2nd column of Table 3.1 for examples). These ratios were selected on the basis of their popularity in the literature and their potential relevancy to the study. Furthermore, Altman included some new variables to his analysis which were not mentioned by previous studies. From the original list of 22 variables, five

were selected as best predictors of corporate bankruptcy. Worthily to mention, the selected variables did not contain all of the most significant variables measured independently. As Altman (1968) stated, "the contribution of the entire profile is evaluated and, since this process is essentially iterative, there is no claim regarding the optimality of the resulting discriminant function." The final function, however, does the best job among the alternatives after numerous computer runs with different ratio profiles, and is as follows: $Z = 0.012x_1 + 0.014x_2 + 0.033x_3 + 0.006x_4 + 0.999x_5$. The description of the final ratios are shown in Table 3.1.

Ratio	**Variables (ratios)**	**Description (Altman, 1968)**
x_1	Working capital/total assets	A measure of net liquid assets of the firm relative to the total capitalization.
x_2	Retained earnings/total assets	Retained earnings is the account which reports the total amount of reinvested earnings and/or losses of a firm over its entire life.
x_3	Earnings before interest and taxes/total assets	A measure of true productivity of the firm's assets, independent of any tax or leverage factors.
x_4	Market value equity/book value of total liabilities	Equity is measured by the combined market value of all shares of stock, preferred and common, while liabilities include both current and long term. The measure shows how much the firm's assets can decline in value before the liabilities exceed the assets and the firm becomes insolvent.
x_5	Sales/total assets	Known as capital-turnover ratio and is a standard financial ratio illustrating the sales generating ability of the firm's assets.

Table 3.1: Description of variables used by Altman's Z-score

Caution must be taken when using the formula. Variables x_1 to x_4 must be calculated as absolute percentage values, e.g. a firm with 10% working capital/total assets should be included as 10.0 rather than 0.10. Furthermore, variable x_5 should be expressed in a different manner: sales/total assets ratio of 200% should be included as 2.0 rather than 200. Several other studies have found a more convenient specification of the model: $Z = 1.2x_1 + 1.4x_2 + 3.3x_3 + 0.6x_4 + 1.0x_5$. Only variables x_1 to x_4 are now inserted as the more commonly written percentage notation, e.g. 10% as 0.1.

An F-test can be used to determine the overall discriminating power of the model. It calculates the ratio of the sum of squares between the groups to the sum of squares within the groups. Ideally, we want this to be maximized as it spreads the means of the groups apart and, simultaneously, reduces dispersion of the individual point with respect to its group mean (Field, 2009). Logically, this test is appropriate because the objective of MDA is to identify and utilize variables which best discriminates between groups and, at the same time, are most similar within groups. The F-test for Altman's Z-score for the distressed group is F=20.7 (mean = -0.29), similarly, the non-distressed group is F=3.84 (mean = 5.02). The significance test therefore rejects the null-hypothesis that

the observations come from the same population.

As stated before, the initial sample was examined using data compiled one financial statement prior to bankruptcy. The model classified 95% of the total sample correctly, being extremely accurate. A second test was conducted to observe the model's ability to predict bankruptcy two financial statements (being two years). The model classified 72% of the firms correctly. The accuracy is, however, biased upward due to sampling errors in the original sample and search bias (Altman, 1968). Without going into details, Altman used several validation techniques, such as test and training samples, introducing new samples to test the model, and varying the cutoff point (the point where distressed and non-distressed firms intersect) over various time periods. A classification and prediction accuracy of bankruptcy firms is shown in Table 3.2.

Year(s) prior to failure	Original Sample (33)	Holdout sample (25)	1969-1975 Predictive sample (86)	1976-1995 Predictive sample (110)	1997-1999 Predictive sample (120)
1	94% (88%)	96% (92%)	82% (75%)	85% (78%)	94% (84%)
2	72%	80%	68%	75%	74%
3	48%	-	-	-	-
4	29%	-	-	-	-
5	36%	-	-	-	-

Table 3.2: Z-score classification and prediction accuracy of distressed firms with cutoff point 2.67 (1.81 cutoff in parenthesis)

At this point, we have described Altman's original Z-score as developed in 1968. Table 3.2 shows its predictive power on various samples and remarkable high accuracy and robustness despite its development over 40 years ago. Subsequently, several revised Z-score models have been developed and even a second-generation model known as the ZETA® Credit Risk Model (Altman, Haldeman, & Narayanan, 1977), which contains several enhancements to the original Z-score approach. Despite these evolutions, we were merely concerned with describing the initial model and pointing out the various steps that led to the formula. Describing other improved models at length may even distract the reader from our initial focus, a similar Z-score model for FLOSS projects. We will continue our literature search by discussing variables that characterize distressed FLOSS projects, which we will label as failed FLOSS projects. These variables in turn are relevant to compose our initial sample for further analysis. At this stage, unfortunately, the definition of a failed FLOSS project is not so evident as a distressed firm, where filing for bankruptcy labels the company as such.

3.2 Success and Failure Determinants within Open Source (FLOSS) Projects

The definition and the study of success and failure in the context of open source projects have gained considerable interest from scholars and researchers. An

important question herein lies within the definition of success and failure. What constitute a successful FLOSS project and, similarly, when is a FLOSS project a failure? Despite the overwhelming amount of studies on the subject of project success, little is written about the factors that determine project failure. We believe that this does not necessarily pose a problem, as the lack or absence of any success factor could imply a failure factor. In the next sections we will elaborate on various studies that specifically defined determinants for FLOSS success or failure. Besides that, we will report on the operationalization they used, i.e. the way a determinant was concretely measured.

3.2.1 Success and Failure in Initiation and Growth Stage

English and Schweik (2007) have conducted eight interviews with FLOSS developers to get insights into the independent variables that are important to FLOSS project success and tragedy. By tragedy they refer to the tragedy of the commons, a famous article by Garrett Hardin about how to manage commons appropriately (Garett, 1968). In the sequel, we will continue to use the term failure instead of tragedy, as we believe they constitute the same meaning and help to avoid unnecessary confusion. Rather than focusing solely on project success, they specifically tried to define project failure. Based on their results, they developed a six-class system to describe success and failure of FLOSS projects, which can be seen in Table 3.3. Data was retrieved from the SourceForge.net database on August 2006.

Initiation is defined as the start of a project up to the point of their first release, and *Growth* as the period after this release (Schweik, 2005). Therefore, a project is classified as *Success in the Initiation stage* (SI) when it has produced a first public release. Projects that are abandoned before producing a first public release are classified as *Failure in the Initiation stage* (FI). The word abandoned is, however, subjective and leaves room for potential wrong classification. They define abandoned projects as projects that have few forum posts, few emails to emails lists, no code commits or few other signs of project activity over a time period of one year. In addition, they state that abandonment could be more precisely measured by taking into account the changes in lines of code in the concurrent versioning system (CVS).

CVS enables team members to store code at a central location. This enables members to retrieve the source code to make changes. The CVS keeps track of every change, including what was changed, when it was changed and who made the change. This helps developers to work in parallel and to avoid situations where developers can overwrite each other's work accidentally. A commit occurs when a developer uploads an altered source code file (Midha & Palvia, 2012). Furthermore, little activity on developer email lists and forums during a one-year time period are precursors for abandonment. In general, their analysis indicates that projects in the Initiation phase, that have not had a release for a year, are generally abandoned.

Projects are classified as *Success in the Growth stage* (SG) when it exhibits three useful releases divided over a 6-month time period. Again, useful release is subjective and was measured by taking into account the number of downloads, as this captures the concept of utility. In addition, they emphasize that it could be measured more precisely by including a content analysis on utility of software on data collected from user forums, email archives or web searches. Furthermore,

Class (abbreviation)	**Definition (D) and Operationalization (O)**
Success Initiation (SI)	D: Developers have produced a first release. O: At least 1 release (Note: all projects in the growth stage are SI).
Failure Initiation (FI)	D: Developers have not produced a first release and the project is abandoned. O: 0 releases AND $\geq$ 1 year since SourceForge project registration.
Success Growth (SG)	D: Project has achieved three meaningful releases of the software and the software is deemed useful for at least a few users. O: 3 releases AND $\geq$ 6 months between releases AND does not meet the download criteria for failure detailed in the FG description below.
Failure Growth (FG)	D: Project appears to be abandoned before producing 3 releases of a useful product or has produced three or more releases in less than 6 months and is abandoned. O: 1 or 2 releases and $\geq$ 1 year since the last release at the time of data collection OR <11 downloads during a time period greater than 6 months starting from the date of the first release and ending at the data collection date OR 3 or more releases in less than 6 months and $\geq$ 1 year since the last release.
Indeterminate Initiation (II)	D: Project has no public release but has significant developer activity. O: 0 releases and <1 year since project registration.
Indeterminate Growth (IG)	D: Project has not yet produced three releases but shows development activity or has produced 3 releases or more in less than 6 months and it has been less than 1 year since the last release. O: 1 or 2 releases and <1 year since the last release OR 3 releases and <6 months between releases and <1 year since the last release.

Table 3.3: Six FLOSS success/failure classes and their methods of operationalization (English & Schweik, 2007).

it can be improved by considering a more careful constructed download criteria that take into account the project's lifecycle and downloads from various time periods. Projects are classified as *Failure in the Growth stage* (FG) when they are abandoned before producing three releases or, in cases where three releases have been produced, failed to deliver a useful product.

English and Schweik (2007) extent their classification by adding an indeterminate stage for Initiation and Growth. Projects that show developer activity but have not released a first public release are classified as *Indeterminate in the Initiation Stage* (II). A project that is successful in the initiation phase automatically becomes an indeterminate project in the growth stage. Finally, projects that have not produced three releases but show developer activity, or released three releases over less than six months, are classified as *Indeterminate in the Growth stage* (IG).

English and Schweik (2007) made a first attempt to classify projects in either success or failure for two development stages. Furthermore, they have opera-

tionalized their classification by defining specific parameters to mine projects from SourceForge.net. Such classification is extremely helpful when identifying failed projects for the development of a Z-score for FLOSS projects. Just as Altman (1968) sought for information on distressed firms one year prior to bankruptcy, we need similar information on FLOSS projects, preferable one year prior to becoming a failed project. Therefore, projects in time period t_i, which are classified as failed, can be a success in period t_{i-1}. Similarly, information is needed on projects that are classified as failed in t_{i-1}, but a success in t_{i-2}. Doing so, we can research the defining determinants to predict failure one year prior to the event.

3.2.2 Reclassification of Success and Failure in Initiation and Growth Stage

Wiggins and Crowston (2010) have build upon English and Schweik's first attempt to classify FLOSS projects into success and failure by reclassifying FLOSS projects in a similar manner. Their re-operationalization to extract data was modified in two respects: (1) the success in growth stage had ≥ 3 releases AND ≥ 6 months between most recent and third most recent release AND >10 downloads. (2) The failure in growth stage had 1 or 2 releases AND ≥ 1 year since the most recent release OR 3 or more releases AND ≥ 1 year since most recent release OR ≤ 10 downloads (Wiggins & Crowston, 2010). Another important difference between English and Schweik's original work is the qualification of release rate as an indicator of the sustainability of project activity. The original measure evaluates release rates by whether consecutive releases have at least a 6-month time period between them. Wiggins and Crowston argue that this would privilege older projects rather than more stable projects. Two possible solutions are presented to overcome such misclassification: (1) setting a threshold for the amount of time between the most recent series of releases, rather doing so for all releases (English & Schweik, 2007). (2) Evaluate the average time between each release against a threshold (Wiggins & Crowston, 2010).

Wiggins and Crowston reclassified FLOSS projects by replicating English and Schweik's original work, followed by two variations for measuring release rates as described above. Although their work found slight differences in classification, these are not important for our interest. We are primarily concerned with the parameters for classification, rather than knowing exact percentages. An important commentary from Wiggins and Crowston is that a success remains a success, even if a project becomes inactive. This is a conservative classification choice, reflecting the reality that successful projects may enter a retirement stage after active development stops, but in which it is still useful to others. This is another view than English and Schweik's, where a successful project can be abandoned and fall into disrepair.

3.2.3 Survival Factors in Initiation and Growth Stage

Wang (2012) studied survival factors for FLOSS projects by taking into account the same non-static project lifecycle approach as described by English and Schweik (2007), i.e. the presence of an Initiation and Growth stage. She rationalizes that FLOSS projects evolve from one stage to the next, and what

works at one stage may not necessarily work in another. Thus, her research distinguishes FLOSS projects that are at the initial stage from those at the growth stage, and that the survival factors are subject to change depending on the stage of the project. These survival factors can function as warning indicators and hence be interesting for our FLOSS project failure prediction, i.e. a Z-score for FLOSS projects.

Wang's research was performed using data drawn from SourceForge.net. It consisted of a stratified random sample of 2,220 projects taken from January 2005 to January 2010. The definition of initiation and growth stage was adopted from English and Schweik's work (2007). Wang's analysis showed that survival factors do in fact change when FLOSS projects evolve. In both stages, *user/developer participation* and *service quality* are associated with project survival. It is thus essential to have high quality developers who are devoted to participate in the project development activity and to provide quality service to the users. *License restrictiveness* has a marginal impact on survival at the initiation stage, but has no impact at the growth stage. License restrictiveness relates to the type of license used to redistribute the software. This can vary from very restrictive, e.g. strong-copy-left license that require any subsequent or derivative software to inherit the original license, to less restrictive, e.g. weak-copy-left license where subsequent software can be licensed under a different, or similar license. There are also non-copy-left licenses where subsequent or derivative work is not obliged to inherit the original license (Lerner & Tirole, 2005). FLOSS projects targeted at technical users have, in both initiation and growth stage, a higher likelihood of survival compared to other types of projects. Wang concludes that technically sophisticated users are still the niche market for FLOSS applications (Wang, 2012).

One network property that has been recurrently found to positively affect FLOSS project outcomes is network size (Grewal, Lilien, & Mallapragada, 2006; Wu, Goh, & Tang, 2007). In an affiliation network, a FLOSS project develops an internal network, whose size is captured by the number of developers in the project, and an external network, whose size is captured by the total number of projects in which this particular project's developers are members (Wang, 2012). The sizes of internal and external networks have significant survival effect in the initiation stage, but negligible effect in the growth stage. In contrast, the quality of an external network positively impacts survival in both stages. To conclude, an overview of Wang's findings is shown in Table 3.4, categorized into three classes: developer, software and community characteristics.

Class	Predictor	Definition	Studies	Outcome
Developer characteristic	User and developer participation effort	Level of participation from users and developers	(Mockus, Fielding, & Herbsleb, 2002)	Positively impacts survival rates for both initiation and growth stage
Software characteristic	License restrictiveness	Whether the license of FLOSS software contains highly restrictive terms	(Stewart, Ammeter, & Maruping, 2006)	Highly restrictive licenses have a (marginally) positive effect on survival in the initiation stage, but no effect in growth stage.
	Targeted users	Whether a FLOSS project is targeted at general end-users or tech-savvy end users	(Stewart et al., 2006; Wu et al., 2007)	FLOSS projects targeted at tech-savvy users have a higher likelihood of surviving in both stages compared to other type of FLOSS projects
Community Attributes	Social network ties	The number of direct and indirect ties a FLOSS project has	(Grewal et al., 2006; Wu et al., 2007)	Both internal and external networks have positive survival impact in initiation stage, but no effect in growth stage.
	Quality of social ties	The extent to which a FLOSS project is connected to other important FLOSS projects	(Grewal et al., 2006)	The quality of external network positively impacts survival rates in both stages.

Table 3.4: Overview of survival measures for FLOSS projects categorized into three classes

Wang's results demonstrate that FLOSS project outcomes must be examined in its dynamic context. More specifically, time effects are important when studying survival measures and should not be ignored. Although not fully adopted by all researchers, many have researched survival and/or success factors from a more static point of view. We will continue our literature review section by describing them at length. It is, however, important to incorporate as much initial success or failure variables for the development of our FLOSS Z-score, similar as Altman developed his measure out of an initial set of 22 variables (Altman, 1968).

3.2.4 FLOSS Success in the Context of Traditional Information System Development

Crowston et al. (2006) made a first attempt to develop success measures for FLOSS projects in the context of traditional Information System (IS) development. For that, they looked at DeLone and McLean's Model of IS success

(DeLone & McLean, 1992) and adopted several measures to fit the FLOSS context. Additionally, they have assessed several FLOSS success measures found in literature and commented on their appropriateness and utility. Their work is summarized in a set of possible measures of FLOSS development effectiveness and related operationalization.

DeLone and McLean's IS model consists out of six interrelated measures of success: *system quality*, *information quality*, *use*, *user satisfaction*, *individual impact* and *organizational impact*. Their model was built by considering a process model that has only three components: the creation of a system, the use of a system, and the consequences of a system (Crowston et al., 2006). Table 3.5 shows a summary Crowston's et al. study of concepts for IS success in FLOSS context.

As can be seen from Table 3.5, Crowston et al. (2006) extracted various measures from traditional IS development that can be used in the context of FLOSS. These measures with their potential indicators hence contribute to our initial list of variables for the development of the FLOSS Z-score model. Crowston et al. (2006) also analyzed SourceForge.net data by looking at three potential indicators for success: *developer counts*, *speed of bug fixing* and *popularity*. Again, we are not interested in the classification of the various projects hosted in SourceForge.net, rather it is interesting to know how they operationalized their indicators. In addition, their classification is based on data from 2001 up to 2005, leaving a big gap in time for valid trend analysis.

The number of developers involved in a project was operationalized in two ways. First, SourceForge.net provided developer counts on the project's summary page. Second, they counted the number of individuals who posted a bug report or message to the SourceForge.net bug tracker. The speed of bug fixing was calculated by using the average time span of open and close timestamps recorded by the bug tracker. The popularity of a FLOSS project was measured in three ways. First, the number of downloads and project page views were extracted from SourceForge.net. These in turn were recalculated to downloads and page views per day to compensate for difference in project ages, as some projects were older than others. Another used measure for popularity was whether the project produced programs that were included in the Debian Linux distribution. This not so apparent measure is supported by their argument "In keeping with a portfolio approach to success measurement, we also measured popularity by examining whether the project produced programs that were included in the Debian Linux distribution, the largest distribution of FLOSS. Debian is distributed as a base installation and a set of additional packages for different programs. Not all the programs produced by FLOSS projects are candidates for inclusion in a Linux distribution (for example, some projects are written only for the Windows or Mac OS X platform), so this measurement was taken only for projects producing programs eligible for inclusion in the distribution (Crowston et al., 2006)."

Process phase	Measure	Potential indicator
System creation and maintenance	Activity effort	File releases, CVS check-ins, mailing list discussions, tracker discussions, surveys of time invested
	Attraction and retention of developers (developer satisfaction)	Size, growth and tenure of development team through examination of registration, CVS logs. Posts to dev. mailing lists and trackers. Skill coverage of development team. Surveys of satisfaction and enjoyment
	Advancement of project status	Release numbers or alpha, beta, mature self-assessment, request for enhancements implemented
	Task completion	Time to fix bugs, implementing requests, meeting requirements (e.g. J2EE specification). Time between releases
	Programmer productivity	Lines of code per programmer, surveys of programmer effort
	Development of stable processes and their adoption	Documentation and discussion of processes, rendering of processes into collaborative tools, naming of processes, adoption by other projects/endeavors
System quality	Code quality	Code analysis metrics from software engineering (modularity, correctness, coupling, complexity)
	Manageability	Time to productivity of new developers, amount of code abandonment
	Documentation quality	Use of documentation, user studies and surveys
System use	User Satisfaction	User ratings, opinions on mailing lists, user surveys
	Number of users	Surveys (e.g. Debian popularity contest), downloads, inclusion in distributions, package dependencies, reuse of code
	Interest	Site pageviews, porting of code to other platforms, development of competing products or spin-offs
	Support effectiveness	Number of questions effectively answered, time required to assist newbies
System consequences	Economic implications	Implementation studies, e.g. total cost of ownership, case studies of enablement
	Knowledge creation	Documentation of processes, creation of tools
	Learning by developers	Surveys and learning episode studies
	Future income and opportunities for participants	Longitudinal surveys
	Removal of competitors	Open sourcing (or substantial feature improvement) of competing proprietary applications

Table 3.5: Summary of concepts for Information System success in FLOSS context (Crowston et al., 2006).

Crowston, and others, also studied other possible measures with respect to FLOSS project success that are not specifically related to traditional IS development. These measures include developer activity / effort, the size of the development team, project development status, task completion, interest and copies in circulation (Crowston, Annabi, Howison, & Masango, 2004; Crowston & Scozzi, 2002). Developer activity/effort was operationalized by looking at the SourceForge.net activity level, whereas the size of the development team was operationalized by the number of posters in the bug tracker. Furthermore, the project development status was observed by looking at the development stage as reported by SourceForge.net and task completion as the speed in which bugs were closed. Interest and copies in circulation were operationalized as SourceForge.net page views and downloads, respectively.

Building on the work of Crowston et al. (2006) and DeLone and McLean (1992), S.-Y. Lee et al. (2009) studied OSS success characteristics by looking at traditional Information System development. Their study adapted a new IS success model from DeLone and McLean, which included service quality (DeLone & McLean, 2003). S.-Y. Lee et al. (2009) developed an OSS success model that consists of software quality and community service quality as determinants of user satisfaction and OSS use, which in turn, determine individual net benefits (job and decision making performance). Software quality and community service significantly affected user satisfaction, and OSS use was significantly influenced by software quality and user satisfaction. Furthermore, OSS use and user satisfaction together significantly influence individual net benefits. In contrast to DeLone and McLean's (2003) model, community service quality has no significant effect on OSS use. It is, however, important to note that Lee's et al. study was conducted mostly in Korea and that their results may not be generalizable due to cultural differences. Nevertheless, their determinants can be incorporated in our study and added to the initial list of variables for the development of the FLOSS Z-score model.

3.2.5 FLOSS Success Measures and their Determinants

Midha and Palvia (2012) studied factors affecting the success of open source Software. Their purpose was to understand the impact of various factors, categorized as intrinsic and extrinsic factors, on OSS project success over the first three years of its life. They furthermore distinguished success into technical success and market success. Defining technical success as the degree of contributions to enhance the software, operationalized by the total number of commits made in the source code of the N-1th version. Whereas market success is defined as project popularity, i.e. a high level of interest displayed in the project by its consumers (Midha & Palvia, 2012). Market success was operationalized by using the number of downloads as a surrogate, which was adopted from (Grewal et al., 2006) and (Rai, Lang, & Welker, 2002). Their study include license type, user base, developer base, language translation, responsibility assignment, complexity and modularity as factors influencing either technical or market success. Similar to other studies (English & Schweik, 2007; Wang, 2012; Wiggins & Crowston, 2010), Midha and Palvia (2012) studied factors over different time periods of a project's lifecycle. In contrast to these studies, Midha and Palvia used a four time period classification to represent the four stages noted by Wynn (2003) and Schweik and Semenov (2003). For each project, data was collected

for the first version (T = t1) representing the initiation stage; a version released within 3 months of the first version (T = t2) and another version released within 6 to 9 months of the first version (T = t3) representing intermediate growth stage; and, yet another version within 2-3 years of the first version (T = t4) representing growth stage (Midha & Palvia, 2012). Data was collected from SourceForge.net. An overview of their results is shown in Table 3.6.

Factor	**Description**	**Operationalization**	**Results**
License Type (DV: market and technical success)	The type of license used to comply with Open Source Development	GPL projects (restrictive=1) and others (non-restrictive=2)	License restrictiveness plays a significant role in the project's popularity only for the first versions of the OSS project with respect to market success. OSS projects in t3 / t4 that use a non-restrictive license have lower technical success.
User Base (DV: market success)	The number of users of the project's software	(Cumulative) number of times the project has been downloaded until the N-1th version	OSS projects that have a large user base are more popular in all stages, i.e. leads to more market success.
Developer Base (DV: market and technical success)	The number of developers contributing to the software	(Cumulative) unique number of developers that have contributed to the development of the project until the N-1th version	A high developer base leads to more market success except at t3. The number of developers has a positive impact on technical success only at t2.
Language translation (DV: market success)	The ability to use the software in the user's native language	The number of language translations for each version	OSS projects with a high number of translations are more popular with respect to market success.
Responsibility assignment(DV: technical success)	OSS projects that delegate responsibilities to the teams developers	The fraction of the number of tasks assigned to someone over the total number of tasks listed for N-1th version after its release	Assigning responsibilities to developers increases technical success in all stages
Complexity(DV: technical success)	The structural and algorithmic complexity of the source code	McCabe's cyclomatic complexity measure	OSS projects with high code complexity lead to lower technical success in all stages.
Modularity (DV: technical success)	The way the software is build so developers can work in parallel	The number of modules (group of member functions, including classes, namespaces, and interfaces) in the software	OSS projects with high modularity gain more technical success in all stages.

Table 3.6: Overview of Midha and Palvia's (2012) results to determine success factors for OSS projects.

Comino et al. (2007) studied FLOSS project success by looking at the different stages a project can reside in. They measured success of a FLOSS project in terms of the development stage it has reached (Comino et al., 2007). Their analysis showed that FLOSS projects that are distributed under restrictive licensing terms have a lower probability of reaching a more advanced development stage. The effect of license restrictiveness fades away for more recent projects. License restrictiveness is hence a popular indicator that impacts FLOSS development as several other studies have also studied this measure, e.g. Midha and Palvia (2012); Wang (2012); Subramaniam, Sen, and Nelson (2009). Furthermore, applications for technically more sophisticated users have greater chance of evolving to a more stable release. We have discussed this finding earlier from Wang's research (2012). Sophisticated applications for high end-users are thus more likely to stimulate the contributions from OS developers and, consequently, leading it to a more successful FLOSS project. Another interesting result found was a non-linear relationship between the size of community of developers and the probability of success of a project. Comino et al. (2007) argue that possible coordination problems might emerge when enlarging the group of developers.

Sen, Singh, and Borle (2012) investigated two success measures for FLOSS projects, namely subscriber base (the number of subscribers in a certain time period) and developer base (the number of developers in a certain time period). Similar to a large number of other studies, data was collected on projects from SourceForge.net for which complete information was available, and which had been registered between January 1999 and December 2005. The variables under study included OSS license, operating system, programming language, whether or not the project accepted financial donations, and user type (target audience) (Sen et al., 2012). Their analysis showed that the age of a FLOSS project resulted in an increase in both the number of subscribers and developers. In addition, FLOSS projects that develop software for the Windows/Unix/Linux operating system, and are written in C/C#/C++ attract more subscribers and developers than projects that do not have these characteristics. Another interesting results can be found in FLOSS projects that receive financial donations and are targeted at IS / IT professionals. Sens et al. (2012) analysis revealed that these projects have more subscribers than other FLOSS projects. We have discussed similar results for FLOSS projects targeted at specific groups, e.g. Wang (2012); Comino et al. (2007). As noted by various other studies described in previous subsections, Sen et al. (2012) also found that FLOSS projects that use a semi-restrictive license experience a decrease in the number of subscribers and attract fewer developers.

Subramaniam et al. (2009) investigated open source software success using longitudinal data from SourceForge.net. Their research spanned 5 years to account for any changes found within FLOSS projects and the differences among them. The success measures under study were developer interest, project activity and user interest. Instead of interpreting these factors as independent, Subramaniam et al. (2009) also studied the potential relationships among them. They determined two categories of factors that affect their success measures: time invariant factors (license use, operating system and programming language) and time variant factors (project status, project activity, user interest and developer interest). These results are comparable to other studies we have discussed in preceding sections. However, Subramaniam et al. (2009) showed that the success measures under study are in fact inter-related and affecting each

other. For example, they demonstrated that developer interest has a positive impact on developer activity. Furthermore, the interest levels of OSS participants and the project activity in any given time period affect the project success measures in the subsequent time period (Subramaniam et al., 2009).

3.2.6 Survival Analysis on Future Development of FLOSS Projects

Samoladas et al. (2010) applied survival analysis techniques for estimating the future development of FLOSS projects. In contrast to various other studies, they used data from FLOSSMetrics. FLOSSMetrics consists of data on thousands of FLOSS projects from various forges, e.g. SourceForge and Github. Their study looked at the probability of continuation in the future, by examining project duration, combined with other project characteristics such as application domain and number of committers (Samoladas et al., 2010). Interestingly, the definition of a non-active or abandoned project was operationalized when the project had less than two commits per month. This in contrast to another related study, where abandoned projects are defined as projects with no activity at all (Evangelopoulus, Sidorova, Fotopoulos, & Chengalur-Smith, 2009). To predict survival of a FLOSS project, Samoladas et al. (2010) used Cox regression models to identify prognostic (determining) factors. Their stepwise Cox regression revealed one significant variable, being the number of committers. Samoladas et al. model shows that projects that exist for more than 10 years are difficult to be abandoned. Regarding the probability of survival of FLOSS projects, projects that are older than 5 year have 40% survival chance, and up to 80% when taking into account follow up projects (projects that suddenly have no commits, possibly caused by coordinators that move the project to another location or make use of another source code management system). They furthermore found that certain types of software have higher probability of survival. For example, projects in 'Software Development' and 'Science' category have more chance to evolve, while 'Database' and 'Security' have less chance. A similar effect was also noticed by Sen et al. (2012); Wang (2012); Comino et al. (2007). To conclude, adding a new developer to the project increases its survivability with 15.8%. This is in contrast to Cominos et al. (2007) study, as they found a non-linear relationship between the number of developers and the success of a project.

3.2.7 FLOSS Project Activity Indicators and Classification

Rainer and Gale (2005) provide a preliminary evaluation of the quantity and quality of 50,000 projects hosted on SourceForge.net. The quality evaluation criteria relate to the responsibility of the owner/developer of the respective project, rather than the reflection of the quality of service provided by the SourceForge.net portal (Rainer & Gale, 2005). Their analysis shows that the projects hosted on SourceForge.net during data analysis are not, or have never been active. To measure activity they have distinguished between two types of activity indicators, namely major and minor indicators. Number of commits, numbers of files added to CVS, number of developers, number of forum messages, numbers of forums, number of mailing lists, total number of bugs, total

number of technical support requests, total number of patches and total number of feature requests are considered major project activity indicators. In contrast, number of open bugs, number of open technical support requests, number of open patches and the number of open feature requests are minor project activity indicators. The number of projects that are active across all activity indicators account for less than 1% of the entire sample. Looking at less stringent indicators, implying those projects with at least some code development, account for approximately 11.6% of the sample. In addition, by solely looking at activity indicators, most projects can be classified as being non-active, or having low code and low user activity.

3.3 Definition Failed FLOSS Project

In order to classify FLOSS projects into failed and non-failed projects, we must first have a definition of a failed FLOSS project. Unfortunately, we can not revert to simple measures used by corporations, such as bankruptcy statements. Focusing on the lack of success, in the context of traditional software product success definitions, seem inapplicable (e.g., profit) or nearly impossible to measure for most projects, e.g. market share, user satisfaction, organizational impact (Crowston, Annabi, & Howison, 2003).

Several studies have indicated that project failure or abandonment differs among development stages or life cycles (English & Schweik, 2007; Wiggins & Crowston, 2010; Wang, 2012; Midha & Palvia, 2012). What works in one stage may not work in another stage. English and Schweik (2007) were among the first to propose that FLOSS projects typically go trough two identifiable stages: initial and growth. Initiation Stage is defined as the period before a first public release, generally the initial period of time when team members of a FLOSS project are collaborating on the first release of the core software code. FLOSS project typically commences because one or a small group of like-minded programmers decide to tackle a shared but unfilled personal software challenge. Software development at the initial stage is typically confined within this small tightly knit group and not open to outside developers for feedback and contribution (Wang, 2012). Growth Stage is defined as the period after a first public release. During this stage, the project is able to attract other developers who want to contribute to the code. Also, the priority of the team shifts from focusing solely on the development of the software product to focusing on the growth of both the software product and the user base (English & Schweik, 2007). Figure 3.1 shows the two stages over time.

In terms of predicting FLOSS project failure by utilizing the projects characteristics, availability of data is essential. Projects residing in the initiation phase, having no public release at all, do not contain important indicators such as number of downloads, number of feature requests, number of technical support requests. Composing an initial test sample for further analysis would yield projects containing just several characteristics. Furthermore, several studies have indicated that projects in the initiation phase, that have not produced a first public release within 1 year after registration, are generally abandoned. We therefore focus on predicting FLOSS project failure for projects that at least have produced a first version, i.e. projects residing in the growth stage. Doing so enables us to use characteristics that describe both the software and the

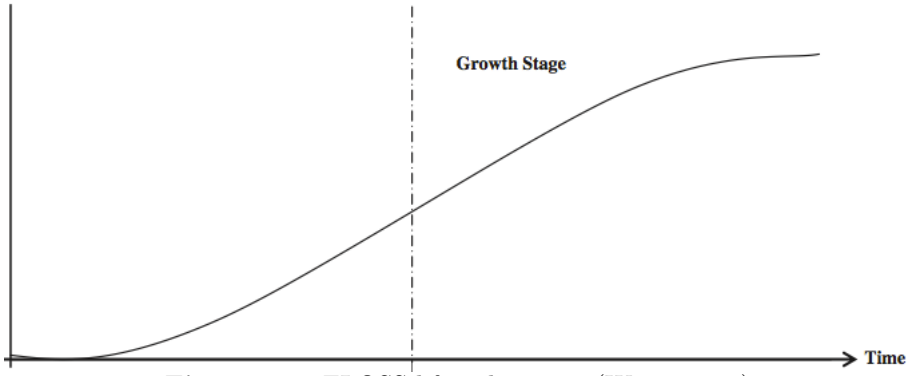


Figure 3.1: FLOSS lifecycle stages (Wang, 2012)

community characteristics.

3.3.1 The Definition

A small number of studies have defined FLOSS project failure and operationalized the definition. We adopted the definition proposed by English and Schweik (2007). They define FLOSS project failure in the growth stage, which they label as tragedy, "A project is considered a Tragedy in the Growth Stage (TG) when it appears to be abandoned without having produced three releases or when it produced three releases but failed to produce a useful software product." They furthermore state that project failure could be more precisely measured by "(1) no code commits or changes in lines of code in the concurrent versioning system (CVS) or other repository over the course of a year, or (2) little or no activity on developer e-mail lists and forums over the course of a year."

The project failed definition as proposed by English and Schweik (2007) can be summarized as:

- 1 or 2 releases and $\geq$ 1 year since the last release at the time of data collection
- $<$ 11 downloads during a time period greater than 6 months starting from the date of the first release and ending at the data collection date
- 3 or more releases in less than 6 months and $\geq$ 1 year since the last release.
- 0 activity in SVN, CVS or Git's versioning system during 12 month period
- $\leq$ 1 forum posts during 12 month period

Projects that do not adhere to the criteria of a failed project are automatically labeled as non-failed. Furthermore, by defining project failure we have answered our sub-research question 2 (S-RQ 2) as stated in Section 2.1.

Chapter 4

Data Preparation

4.1 Relevant Determinants

In Chapter 3 we discussed numerous determinants that may positively or negatively affect FLOSS project outcomes. These determinants serve as the basis for data extraction in subsequent steps of our research. To categorize these determinants we adopted the categorization classes as proposed by Wang (2012). Her categorization is based on three broad classes, namely developer, software, and community characteristics. Developer characteristics refer to the attributes of individual developers affiliated with a particular FLOSS project; software characteristics refer to the attributes related to the software product; and community characteristics refer to the attributes describing the project team and the interaction between the team and its larger social environment. An overview of determinants, derived from our theoretical framework are shown in Table 4.1.

By identifying all determinants that may influence FLOSS project outcomes, we have answered our sub-research question 1 (S-RQ 1) as stated in Section 2.1.

4.2 Availability of Data

The construction of the Z-score model for FLOSS projects is based on the available project metrics as listed on SourceForge.net. This enables project administrators, and other relevant stakeholders, to easily calculate the project's future classification, i.e. failed or non-failed. The availability of projects characteristics (determinants) must be easy to obtain to ease the use of the classification model.

Category	Determinant	Study
Developer Characteristics	-Number of releases	(Schweik, 2005; Crowston et al., 2006)
	-Number of forum posts	(Schweik, 2005; Rainer & Gale, 2005)
	-Number of emails to emails lists	(Schweik, 2005)
	-Number of code commits	(Schweik, 2005; Midha & Palvia, 2012; Samoladas et al., 2010; Rainer & Gale, 2005)
	-Changes LOC in CVS	(Schweik, 2005)
	-Number of developers	(Wang, 2012; Crowston & Scozzi, 2002; Crowston et al., 2004; Midha & Palvia, 2012)
	-Skills of developers	(Wang, 2012; Crowston et al., 2006)
	-Speed of bug fixing	(Crowston & Scozzi, 2002; Crowston et al., 2004, 2006)
	-Number of CVS check-ins	(Crowston et al., 2006)
	-Number of mailing list discussions	(Crowston et al., 2006; Rainer & Gale, 2005)
	-Time between releases	(Crowston et al., 2006; Wiggins & Crowston, 2010)
	–LOC per programmer	(Crowston et al., 2006)
	-Number of posters in bug tracker	(Crowston & Scozzi, 2002; Crowston et al., 2004)
	-Responsibility assignment	(Midha & Palvia, 2012)
	-Code complexity	(Midha & Palvia, 2012)
	-Modularity	(Midha & Palvia, 2012)
	-Total number of bug reports	(Crowston et al., 2006; Rainer & Gale, 2005)
	-Total number of technical support requests	(Rainer & Gale, 2005)
	-Total number of patches	(Rainer & Gale, 2005)
	-Total number of feature requests	(Rainer & Gale, 2005)
	-Number of open bug reports	(Rainer & Gale, 2005)
	-Number of open technical support requests	(Rainer & Gale, 2005)
	-Number of open patches	(Rainer & Gale, 2005)
	-Number of open feature requests	(Rainer & Gale, 2005)
Software Characteristics	-Type of license (license restrictiveness)	(Wang, 2012; Stewart et al., 2006; Midha & Palvia, 2012; Comino et al., 2007; Subramaniam et al., 2009; Sen et al., 2012)
	-Targeted audience	(Wang, 2012; Stewart et al., 2006; Wu et al., 2007; Comino et al., 2007; Sen et al., 2012)
	-Supported platforms/OS	(Crowston et al., 2006; Sen et al., 2012; Subramaniam et al., 2009)
	-Code quality	(Crowston et al., 2006; S.-Y. Lee et al., 2009)
	-Documentation availability	(Crowston et al., 2006)
	-Supported languages	(Midha & Palvia, 2012)
	-Programming language	(Sen et al., 2012; Subramaniam et al., 2009)
	-Age of software	(Sen et al., 2012)
	-Category	(Samoladas et al., 2010)
Community Characteristics	-Number of downloads	(Schweik, 2005; Wang, 2012; Crowston et al., 2006; Mockus et al., 2002; Crowston & Scozzi, 2002; Crowston et al., 2004; Rai et al., 2002; Midha & Palvia, 2012; Sen et al., 2012)
	-Number of web searches	(Schweik, 2005)
	-Number of External networks	(Wang, 2012; Grewal et al., 2006; Wu et al., 2007)
	-Quality of social ties	(Wang, 2012; Grewal et al., 2006)
	-Number of project page views	(Crowston & Scozzi, 2002; Crowston et al., 2004, 2006)
	-User satisfaction/ratings	(S.-Y. Lee et al., 2009; Crowston et al., 2006)
	-Number of forums	(Rainer & Gale, 2005)
	-Accept financial donations	(Sen et al., 2012)

Table 4.1: Overview of determinants that may affect FLOSS project outcomes

As our main objective is to construct a model that enables prediction, the use of retrospective data is essential. Performing data mining on the SourceForge website "as is" provides metrics that are stored on the date of extraction, which in our case are non relevant. For historical data, we should focus on the availability of project data delivered by various SourceForge APIs, as they enable us to specify a date range when collecting data. Additionally, SourceForge provides a monthly dump of their data to the University of Notre Dame which is stored in the so called SourceForge Research Data Archive (SRDA). This data is made available to the academic and scholarly research community under a sublicense from SourceForge.net.

The determinants extracted from our Theoretical Framework (Table 4.1) need to be assessed based on the availability of:

- Historical data provided by SourceForge APIs[1]
- Data available in the SourceForge Research Archive
- Completeness of data starting from January 2011 up to January 2013
- The data needs to be easily obtainable and should contain sufficient valid data points to construct a representative sample.

4.2.1 Availability of Determinants

This sub-section of the paper provides a detailed description of the determinants and the way they are operationalized. Table 4.1 serves as input for the assessment of available data that needs to be extracted for further analysis.

Number of releases The SRDA provides data on the number of releases in the *frs_ schema*. The data, however, contains no values after July 2009, making data unusable for current years. Fortunately, SourceForge provides an API to obtain information on file releases. A release is defined as an increment in the version numbering with file sizes larger than 0 bytes. The number of releases was extracted during a 12 month time period.

Number of forum posts Data on the number of forum posts are stored in the SRDA. However, scanning this data revealed lots of inconsistencies with actual forum posts on SourceForge. We therefore reverted back to the original data provided by SourceForge's Forum API[2]. This API was used to extract data on the number of forums posts during a 12 month time period.

Number of developers Data on the number of developers was collected from the SRDA in the *user_groups* table, which connects users to projects (groups) due to the many-to-many relationships (n:m). Each project was given a non-negative integer for the number of developers.

[2] e.g. https://sourceforge.net/api/post/index/forum-id/526557/rss

Number of CVS check-ins The SRDA provides CVS data but, unfortunately, seems incomplete in several respects. The glitch in the data was also reported on the SRDA wiki. Moreover, we did not find any API that could be used to extract versioning control data. Though we believed this determinant to be important, we extracted data from the SourceForge project pages by spidering the data ourselves.

Total number of bugs/ technical/ feature/ patch requests The SRDA provides data on the total number of bug reports, technical support requests, feature requests and the number of patches. Data was extracted from the SRDA *artifact schema*. These determinants were operationalized by adding all open/-closed/deleted and pending requests for bug reports/tech requests/ feature requests and patch requests during a time period of 12 months.

Total number of open bugs/ technical/ feature/ patch requests Similar to the total number of bug/ tech / feature and patch requests, the number of open requests were extracted from the SRDA *artifact schema*. We counted only requests which were labeled as "open". These determinants were operationalized by counting the number of open bug reports/tech requests/ feature requests and patch requests during a time period of 12 months.

Type of license The SRDA stores information on the project's license type in the *trove_group_link* table. Unfortunately, data is stored as nominal values and projects can contain multiple licenses. Examples of licenses are: Mozilla Public License 1.1 (MPL 1.1); GNU General Public License version 2.0 (GPLv2); MIT License; Academic Free License (AFL). A total of 85 different licenses are available to characterize the project. Taking the type of license into account for further analysis proved difficult, especially when projects can contain multiple licenses. Therefore, we operationalized this determinant by counting the number of distinct licenses a project contains.

Targeted audience The SRDA stores information on 29 different audiences in their *trove_group_link* table. A project can target its software to e.g. developers, researchers. Furthermore, projects can contain one or multiple audiences, making classification troublesome. Therefore, we operationalized this determinant by counting the number of distinct audiences.

Supported languages This determinant was extracted from the SRDA in the *trove_group_link* table. A total of 66 different languages are supported, examples are English, Chines, Spanish, Dutch and Greek. We operationalized this determinant by counting the total number of distinct languages that a project supports, i.e. the oral/written language for the development and use of the software.

Programming language This determinant was extracted from the SRDA in the *trove_group_link* table. A total of 105 different programming languages are available, examples are C#, ASP.NET, Curl and Ruby. We operationalized this determinant by counting the total number of distinct programming languages the project is written in.

Age of software The SRDA stores information on the project's age in the *groups* table. A unix timestamp is used to store the register date of the project. This determinant was operationalized by taking the difference in time between date of the data dump and the register date of the project. The difference between the unix timestamps was recalculated into difference in years, creating a value for the project's age in years. For example, 1.5 stands for 18 months.

Category The SRDA stores information on the type of category the project belongs to. A total of 366 different categories are applicable, examples are 3D Modeling, Algorithms, Collaborative development tools, Medical/Healthcare and Web Services. Projects can reside in multiple categories. This determinant was operationalized by counting the number of distinct categories the project resides in.

Number of downloads Various studies have identified this determinant to be an important indicator for project use (Schweik, 2005; Wang, 2012; Crowston et al., 2006; Mockus et al., 2002; Crowston & Scozzi, 2002; Crowston et al., 2004; Rai et al., 2002; Midha & Palvia, 2012; Sen et al., 2012). Data stored on the SRDA allows extraction on the number of downloads. Furthermore, even downloads during a certain time period can be extracted by subtracting different monthly dumps. However, data seems incomplete and not consistent with data provided by the SourceForge website. Fortunately, SourceForge provides an API to extract download counts for each project. We programmed several scripts to extract download counts from SourceForge APIs. Monthly download counts were accumulated to yearly downloads.

Number of project page views The SRDA stores no information on the number of page views or another representative indicator for web traffic. Fortunately, the SourceForge website provides data on page views which are accessible by mining the project pages. Project administrators are asked to incorporate the SF logo on project's website. Doing so enables SourceForge to keep track of the number of logo requests, i.e. to determine the web traffic of a project outside the SourceForge repository. Furthermore, SourceForge also keeps track of web traffic from the project's repository page. We programmed several scripts to extract information on web traffic data. Data of each project were extracted for a time period of 12 months.

Financial donations Data on financial donations are stored as a binary value in the SRDA. However, we were unable to extract the amount of financial donations and use this as a scale variable. We therefore operationalized this data by using a dummy variable: 0 for no donations, 1 for having received some sort of financial donation.

4.2.2 Discarded Determinants

Several determinants extracted from our Theoretical Framework were not easily captured, not available at all, or have insufficient data points from Sourceforge's APIs or the SRDA. These determinants were discarded from our analysis. We list these determinants in Table 4.2. The determinant is assessed based on the

availability of data on the SRDA or SourceForge API. A checkmark is placed if data was available, and an x-mark was placed when data was not available. The right two columns indicate, when data was available, why the determinant was discarded.

Determinant	Available SRDA	Available SF API	Quality of data	Easily captured
Number of emails to emails list	✗	✗		
Number of code commits	✗	✗		
Changes LOC in CVS	✗	✗		
Skills of developers	✓	✗	-	
Speed of bug fixing	✓	✗	-	
Number of mailing list discussions	✗	✗		
LOC per programmer	✗	✗		
Number of posters in bug tracker	✓	✗		-
Number of forums	✓	✗	-	
Responsibility assignment	✗	✗		
Code complexity	✗	✗		
Modularity	✗	✗		
Supported OS	✓	✗	-	
Documentation available	✓	✗	-	
Number of external networks	✗	✗		
Quality of social ties	✗	✗		
User satisfaction	✗	✗		

Table 4.2: Discarded determinants

No data on Lines Of Code was available from the SRDA or SourceForge APIs. SourceForge did provide an API for file releases and file sizes. We believed that file size would not be a representative indicator for the LOCs, as adding graphical or audio/video to a project would increase its size considerable without a significant difference in LOCs. The same applies to LOC per programmer.

The determinant *Speed of bug fixing* could be extracted from the SRDA by averaging the open and close timestamps of bug reports. Unfortunately, doing so would yield too few data points for further analysis.

The SRDA provides data on bug reports of projects in the *artifact schema*. However, counting the number of posters seems to be a troublesome task. The query request did not provide any information but hanged instead. Moreover, a limited query, or a query for a single project, timed out after a period of time. The SRDA failed to provide any information on this determinant, therefore, we discarded this determinant.

The SRDA stores data on the number of forums used for each projects. Scanning this data shows that approximately 99% of all projects contain 2 forums. The meaning of this determinants therefore seemed irrelevant.

Data on supported platforms is stored in the SRDA in the *trove_group_link* table. A total of 83 different operating systems are applicable to characterize the project. Unfortunately, there exists a value "OS Independent (Written in an interpreted language)", making data hard to interpret when counting the number of different supported platforms. For example, supporting 5 different platforms may not necessarily imply software that can be used on more platforms

than a project that lists a 1 for supported platforms. We therefore discarded this determinant from further analysis.

Data on available documentation and other related information is stored in the SRDA. This data, however, is incomplete as there is no data available after 19th October 2009. Our analysis is focused on data from current years, therefore, we discarded this determinant. Furthermore, we were unable to easily extract documentation data from SourceForge APIs.

4.3 Overview of Determinants

Determinants that are used for analysis are selected based on their availability and the selecting criteria as defined in Section 4.2. We extracted data from the SourceForge Research Archive and from APIs provided by SourceForge. Because the Z-score model for FLOSS projects is used for prediction, the availability of historical data is essential. For that reason, the monthly dumps of SourceForge data provided to the University of Notre Dame was our main point for data extraction. In all cases were this data does not suffice, we reverted to data provided by SourceForge APIs. These APIs enable us to extract data by indicating a time region, e.g. data from January 2012 to January 2013. Table 4.3 provides an overview of determinants for which we found valid and sufficient data. A checkmark is placed to indicate from which source we extracted the data: SRDA or SourceForge API. Additionally, we provided the name of the table/schema or API name.

Determinant	**SRDA**	**SF API**	**Schema/Table/API**
Number of releases		✓	File releases API
Number of forum posts	✓		Forum API
Number of developers	✓		*user_groups*
Number of CVS check-ins		✓	SCM activity on SF
Time between releases		✓	File releases API
Total number of bug reports	✓		*artifact*
Total number of tech. support requests	✓		*artifact*
Total number of patches	✓		*artifact*
Total number of feature requests	✓		*artifact*
Number of open bugs	✓		*artifact*
Number of open tech. support requests	✓		*artifact*
Number of open patches	✓		*artifact*
Number of open feature requests	✓		*artifact*
License type	✓		*trove_group_link*
Targeted audience	✓		*trove_group_link*
Supported languages	✓		*trove_group_link*
Programming language	✓		*trove_group_link*
Age of software	✓		*groups*
Category	✓		*trove_group_link*
Number of downloads		✓	Downloads API
Number of project page views		✓	Traffic activity on SF
Financial donations	✓		*groups*

Table 4.3: Determinants selected for further analysis

4.4 Extra Determinants

Apart from the determinants listed in our Theoretical Framework, we included several other indicators that represent some form of project activity. These determinants were extracted when mining SourceForge APIs, SourceForge's project pages, or found to be interesting when creating queries to extract data from the SRDA. The following indicators were added to our analysis:

Tracker activity Opened and closed tracker activity is stored on the project pages hosted on SourceForge. We extracted bug reports, support requests, feature requests and patches from the SRDA, but have extracted tracker information directly from SourceForge. This determinant was labeled tracker activity and contains, apart from the standard trackers, custom trackers to meet the information management needs of the project. We accumulated the opened and closed trackers for a time period of 12 months.

Number of news items SourceForge provides an API[3] to extract information on news items. We used this information to compute the number of news items a project released during a 12 month time period.

Number of news item posters The data on news items contained email addresses on who posted the message. We used this information to calculate the number of distinct posters during a 12 month time period.

Number of Runtime dependencies Some projects needs certain runtime environments to operate correctly. This can for instance be graphical user interface, a certain tool-kit, a plug-in or webkit. Because this data was available in the SRDA, we extracted the number of runtime dependencies and counted the distinct environments.

[3] e.g. https://sourceforge.net/api/news/index/project-id/156708/rss

Chapter 5

Data Extraction

This chapter describes the data extraction methods we performed to collect data on FLOSS projects from the SourceForge Research Database, the SourceForge APIs, and the project pages from SourceForge.

5.1 Sample Creation

Our initial focus was to extract data from the SourceForge Research Archive (SRDA), as it enabled us to extract data from various monthly dumps. A schematic overview of our data collection phase using the SRDA is depicted in Figure 5.1. We extracted data from projects that were registered on SourceForge.net between January 1st 2011 and January 1st 2012. A total of 523,522 project IDs (SRDA and SourceForge.net label them as group_IDs) were extracted. A large part of these projects contain no data or insufficient data to work with. As a prerequisite, we continued with projects that have listed the following on their project page:

- Type of audience
- Type of category
- Number of developers
- Type of license
- Type of programming language
- Type of runtime dependency

Joining (i.e. using an inner-join) the data on these determinants resulted in a total of 90,683 projects. The following determinants were extracted from the SRDA *trove_group_link* table for all 90,683 projects.

- Number of open bug reports
- Number of open feature requests
- Number of open patches
- Number of open tech requests
- Total number of bug reports
- Total number of feature requests
- Total number of patches

- Total number of tech requests

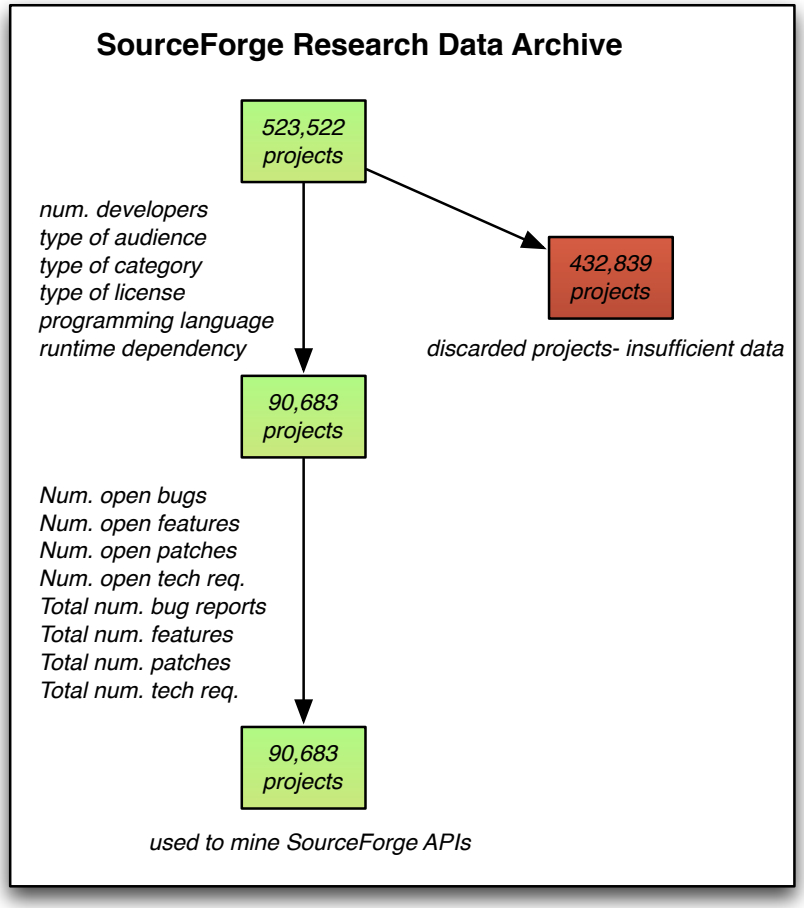


Figure 5.1: Schematic view of the data collection phase using SRDA

SourceForge provides several APIs to extract data directly for each project. A database was constructed that consisted out of all 90,683 project IDs (the result of our previous table join). We used the following APIs to extract our remaining determinants: *File release API*, *Forum API*, *Downloads API* and *Traffic API*. We programmed several scripts that used the project IDs, as stored in the database, and created a loop that used Curl to extract XML data, as provided by the API. This data was then parsed into an array, which contained the elements as provided by the API. We used PHP to extract only those elements which were relevant and stored them in a database containing the other determinants we got from querying the SRDA. Data on the following were stored:

- Number of forum posts
- Number of open and closed trackers (accumulated to tracker activity)

- Number of downloads
- Number of releases
- Average time between releases
- Number of news items
- Number of different news posters

Unfortunately we had to discard the determinant *average time between releases*. When scanning the data we found some problems interpreting the data. This was caused by several reasons. First, projects that produced a single release had no average time between releases. We need to have at least two timestamps to calculate the average time. Second, projects that released e.g. 4 releases in a small period of time would yield a better average time between releases that projects than for instance release monthly. Our calculation for average time between releases (averaging between the youngest and oldest timestamps) was therefore not usable. Moreover, calculating average time between releases by dividing the number of releases over a 12-month period would yield, in statistical sense, not more information than solely using the number of releases.

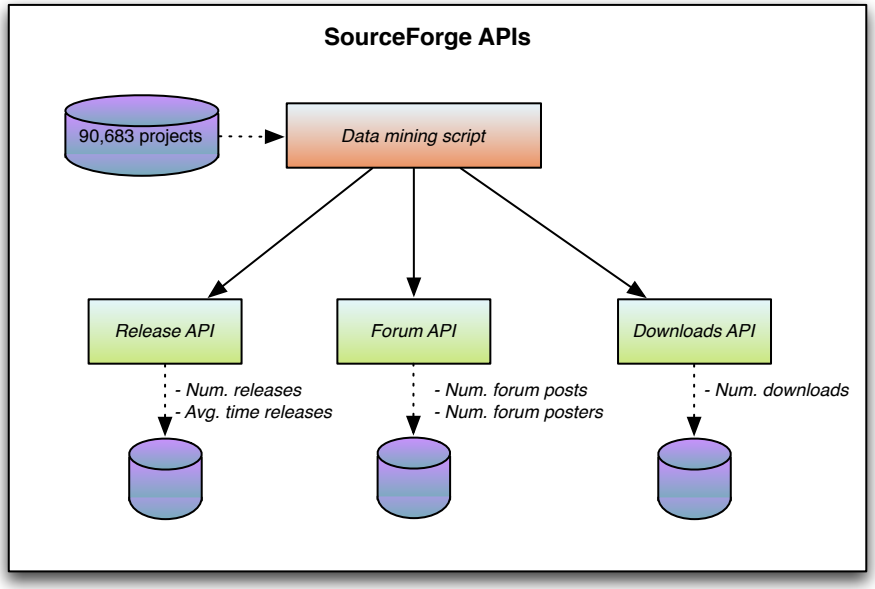


Figure 5.2: Schematic view of the data collection phase using SourceForge APIs

An important indicator for project activity is the number of CVS checkins and the number of project page views. SourceForge provides this data on the project pages by checking the project statistics. We believed this indicator to be important and therefore spidered the project pages to extract this data. By checking out the HTML source code of the project page we were able to receive JSON data on various CVS metrics and project page views. We programmed scripts to store this data into a database. SourceForge labels statistics on versioning systems as SCM activity. Data on SVN, CVS and GITrepository were

stored for both the read and the write transactions. Read transactions consisted out of *CVS anonymous read*, *CVS developer read*, *Git anonymous read*, *Git developer read* and *SVN read txn*. Write transactions consisted out of *CVS write*, *Git write*, *SVN write txn* and *SVN write files*. We accumulated all the CVS, SVN and GitRepository write transactions and labeled them *versioning control write*. The same was done for read transactions which were labeled as *versioning control read*. An example of JSON data where SourceForge shows CVS data is displayed here. For example, anonymous read activity is shown for a certain project. The first number indicates the (unix) time stamp together with the activity count.

Listing 5.1: Excerpt of SourceForge JSON data on CVS activity

```
{
  "data": {
    "anon_read": [
      [1277942400000.0, 20],
      [1293840000000.0, 3],
      [1296518400000.0, 1],
      [1298937600000.0, 7],
      [1301616000000.0, 6],
      [1304208000000.0, 3],
    ],
    "write": [
      [1230768000000.0, 2],
      [1233446400000.0, 1],
    ],
    "dev_read": [
      [1325376000000.0, 1]
    ]
  }
}
```

The data collection process started in May 2013 and took several days. Due to the fact that SourceForge blocks ip addresses after a number of requests have been sent, we were blocked several times during our collection phase. To overcome this, or reduce the time lost when blocked, we used several VPS (Virtual Private Servers) and run several scripts from multiple ip addresses. A schematic view of our data collection set-up is depicted in Figure 5.3. Because we collected data all at once, meaning retrieving data on various determinants per project, we were able to collect 20,000 records daily.

At this point, our database contained data on several determinants of all 90,683 projects. Although a large part of the projects have been filtered out by solely looking at projects that at least have selected their target audience, operating systems etc., a large part of these projects can be considered to be inactive, abandoned or still in initiation phase. To make predictions whether or not a project is destined to be failed or non-failed (see section 3.3.1 for the definition of failed and non-failed projects), we looked at projects that are at least active, are in the growth stage, show some sort of development activity and more importantly, adhere to the definition of being non-failed. These are projects that have produced a public release, show development activity, have

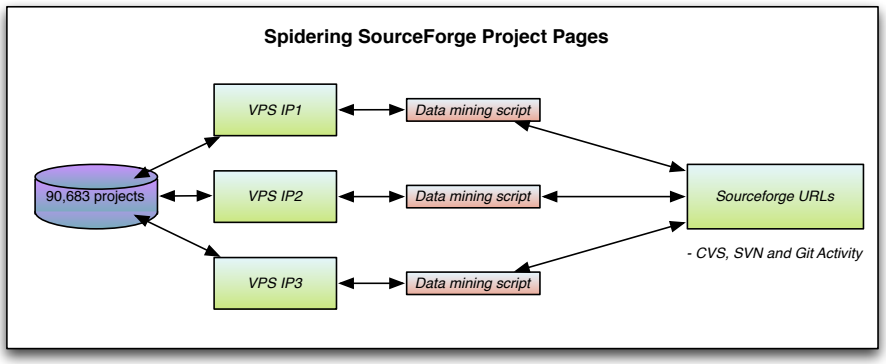


Figure 5.3: Schematic view of the data collection phase extracting data from SourceForge project pages

at least been downloaded once, and adhere to the definition of being non-failed as proposed in Section 3.3.1. A total of 1,447 projects satisfy these criteria. For illustrative purposes we depicted our sample creation method in Figure 5.4.

Another import of aspect of filtering on these criteria is to avoid anomalies in the data. For example, a project can be hosted on SourceForge, have valid data on projects characteristics but show no downloads or releases. This can for instance be seen from the *squirrelmail* project, where it shows substantial tracker and versioning system control activity, but have 0 downloads.

5.2 Classification

We used a time interval of 12 months to create our base sample, which was used to perform multiple discriminant analysis on. More specifically, we used data from January 1st 2011 to January 1st 2012 to create our sample. In order to make predictions, we mined date from January 1st 2012 to January 1st 2013 and searched for projects that were characterized as failed and non failed and labeled them in our initial sample as such. Doing so, we created a dichotomous variable in our base sample where projects that are going to be classified (the following year) are labeled as 0, and projects that were classified as non-failed were labeled as 1. This sample serves as input for performing MDA. Our classification method is depicted in Figure 5.5 for clarification purposes.

5.3 Used Determinants

After querying the SRDA, extracting data from SourceForge APIs and spidering project pages hosted on SourceForge, our initial list of determinants are altered or operationalized in a different manner. Because we were merely concerned of finding determinants which are measured on a scale level, this affected some of the determinants. For example, one of our initial determinants was *type of category*. As projects can contain multiple categories, and the number of different categories was quite large, it was troublesome to create nominal values.

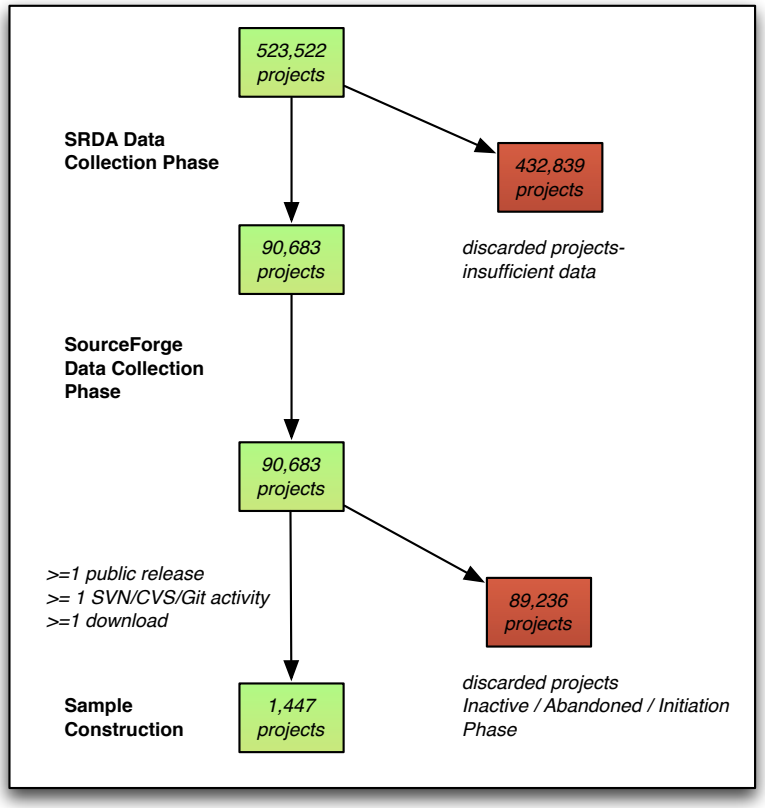


Figure 5.4: Sample creation method

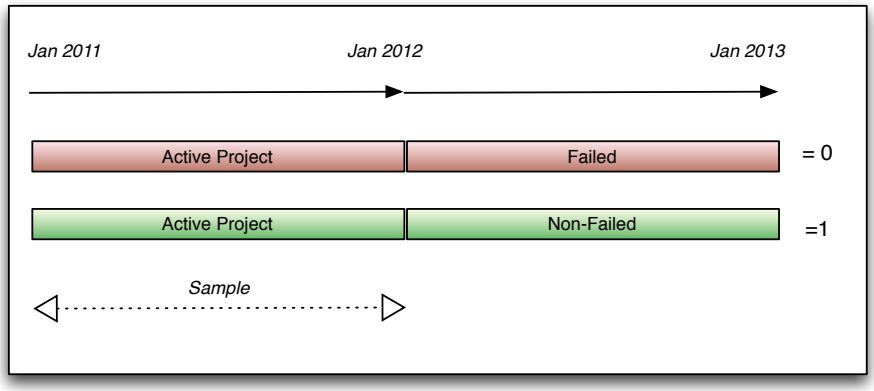


Figure 5.5: Classification method to label projects within our sample as 0 = failed and 1 = non-failed in subsequent year

As a result, we counted the number of categories and used that as a scale variable. At the end, we ended up with the following variables:

1. Number of downloads
2. Number of releases
3. Age of project (years)
4. Number of forum posts
5. Tracker activity (accumulated from open and closed tracker activity)
6. Traffic of SourceForge logo
7. Traffic hits
8. Versioning control read transactions (accumulated from CVS anonymous read, CVS developer read, Git anonymous read, Git developer read and SVN read txn)
9. Versioning control write transactions (accumulated from CVS write, Git write, SVN write txn and SVN write files)
10. Number of news items
11. Number of news posters
12. Number of open bugs
13. Number of open features
14. Number of open patches
15. Number of open tech requests
16. Total number of open bugs
17. Total number of open features
18. Total number of open patches
19. Total number of open tech requests
20. Number of audiences
21. Number of categories
22. Number of developers
23. Number of licenses
24. Number of programming languages
25. Number of runtime dependencies

Chapter 6

Analysis

In previous chapters we have selected relevant determinants that may positively or negatively affect FLOSS project outcomes. We have collected data on these determinants from the SourceForge Research Archive (SRDA), or have mined SourceForge APIs or project pages to collect other essential data that were not stored in the SRDA. Based on this data, we created a sample that consisted out of projects that were hosted on SourceForge between January 1st 2011 to January 1st 2012. A total of 90,683 projects with corresponding data was stored. Projects that were in their growth stage and show activity in downloads and versioning control system were selected for further analysis, 1,447 projects satisfied these criteria. We have adopted the project failure definition from English and Schweik (2007), and improved the definition as defined in Section 3.3.1. Projects were labeled as failed (0 as dummy variable), if in subsequent year, being January 1st 2012 to January 1st 2013, they adhere to the definition of being failed. All other projects were labeled as non-failed (1 as dummy variable).

This chapter describes the process of statistical analysis, and more specifically, how we applied Multiple Discriminant Analysis on our dataset.

6.1 Descriptives

Before proceeding to the main part of our analysis, Table A.1 shows the descriptives of statistics of our complete sample. Determinants that we included in our sample are listed for all 1,447 FLOSS projects. We see large ranges for the determinants *traffic SF logo*, *traffic hits* and *number of downloads*. These determinants vary as they constitute activity from the developers side, as well as the community side, that can easily grow for more successful projects. The descriptive statistics for the failed group are shown in Table A.2, and for the non-failed group in Table A.3.

6.2 Multiple Discriminant Analysis

We have discussed the application of Multiple Discriminant Analysis (MDA), or Discriminant Analysis (DA) in short when explaining Altman's Z-score in Section 3.1. In contrast to multiple linear regression, MDA is able to predict an outcome when the dependent variable appears in categorical form, rather than

being on interval level. This statistical regression technique, therefore, suits our needs best as we have failed and non-failed groups.

MDA involves the determination of a linear equation that will predict which group the case belongs to. The form of the equation or function in its general form is:

$$D = v_1x_1 + v_2x_2 + ... + v_ix_i + a.$$

Where: D: Discriminant function
v: The discriminant coefficient or weight for that variable
x: Respondent's score for that variable
a: A constant
i: The number of predictor variables

The v's are unstandardized discriminant coefficients analogous the the b's in a regression equation. The coefficients try to maximize the distance between the means of the dependent variable. For best results, the function must have strong discriminating power. Once a function has been created from a sample, any new cases can then be classified. The number of discriminant functions is one less than the number of groups. In our case, we have two groups which results in a single discriminant function.

6.2.1 Assumptions

In order to correctly apply MDA, several assumptions of the data must be met. The assumptions are as follows:

- All observations come from a random sample
- Each predictor variable must be normally distributed
- Each of the allocations for the dependent categories in the initial sample/-classification are correctly classified
- At least two or more groups must be defined, with each case belonging to only one group so that the groups are mutually exclusive and collectively exhaustive, i.e. all cases can be placed in a group
- Groups must be defined before collecting the data
- Group sizes of the dependent variable must not be grossly different and should be at least five times the number of independent variables.

The collected data show in various cases highly skewed data, for instance by looking at the number of developers. The assumption that the data must be normally distributed has therefore been violated. However, violations of the normality assumption are usually not "fatal", meaning that the resultant significance tests etc. are still "trustworthy". In addition, we have tried several methods to rectify the skewness of the data. A popular and often used method to transform non-normal distributed data into normal distributed data is the Box-Cox transformation (Box & Cox, 1964). We have applied this transformation technique to our dataset to improve the basis of MDA. Unfortunately, the Box-Cox method seem to have no effect on the distribution of the data. Other transformation techniques, such as log and square-root transformation also have negligible effect on the distribution of the data.

The minimum ratio of valid cases to independent variables for discriminant analysis is 5 to 1, with a preferred ratio of 20 to 1. We have 26 independent

variables and 1,447 cases, resulting in a ratio of approximately 55 to 1. It thus satisfies the minimum as well as the preferred ratio. In addition to the requirement for the ratio of cases to independent variables, discriminant analysis requires that there be a minimum number of cases in the smallest group defined by the dependent variable. The number of cases in the smallest group must be larger than the number of independent variables, and preferably contain 20 or more cases. We have 517 cases in the failed group and 930 cases in the non-failed group. The number of cases in the smallest group, being 517, is more than the number of independent variable, satisfying the minimum requirement. In addition, the number of cases in the smallest group satisfies the preferred minimum of 20 cases.

6.2.2 Stepwise Analysis

When using stepwise discriminant function analysis, a discriminating model is built step-by-step. More specifically, at each step all variables are reviewed and evaluated to determine which one will contribute most to the discrimination between groups. That variable will then be included in the model, and the process starts again. This in contrast to using all variables at once. As we want to find the best set of predictors, a stepwise method suits our specific needs best. Moreover, a stepwise method is often used in an exploratory situations to identify those variables that might later be used for a more rigorous theoretically driven study. The selection of variables to be entered in the analysis is based on the *F to remove* and *F to enter criteria*. The F value for a variable indicates its statistical significance in the discrimination between groups, that is, it is a measure of the extent to which a variable makes a unique contribution to the prediction of group membership.

6.3 Results

We used the statistical software package SPSS to perform MDA. This section elaborates on the output tables and analysis of the results.

By looking at mean scores of the variables between the two groups we can infer some sort of difference. Big differences between means are usually indicators for good discriminating variables. We would prefer, however, statistical evidence in this matter. Table 6.1 shows the result of a simple F-test between the mean scores for each variable between the two groups (failed and non-failed classification). The table shows that several variables, when taking $\alpha = 0.05$, to have a significant difference between groups. The variables are *number of categories*, *number of licenses*, *number of programming languages*, *versioning control write transaction*, *number of releases*, *average time between releases*, *number of news items*, and the *age of a project*. The smaller the Wilk's Lambda is, the more important the variable is to the discriminant function.

Table A.4 shows the pooled within groups matrix. This matrix shows the correlation between the variables taking into account when performing MDA. The matrix must be inspected for multicollinearity. Multicollinearity occurs when one independent variable is so strongly correlated with one or more other variables that its relationship to the dependent variable is likely to be misinterpreted. Its potential unique contribution to explaining the dependent variable is

minimized by its strong relationship to other independent variables. The correlation matrix shows high correlations among the variables *total number of bugs* in relation with *number of open bugs* ($r = 0.988$), *total number of features* in relation with *number of open features* ($r = 0.846$), *total number of patches* in relation with *number of open patches* ($r = 0.99$). We have tried to adjust the variables that represent open patches, open bugs, open features and open tech requests by converting them as percentages of the total number. Due to the fact that a large part of the projects contain 0 total and 0 open e.g. bugs (possibly causing the high correlation coefficient), the division ends up with a *null* value. Furthermore, the high correlation could also be caused as one variable basically is a sub-set of another variable. When performing MDA, SPSS will exclude these cases, leaving us with too few cases to continue. Numerous runs by excluding several variables resulted in no improvements in terms of correct classification results. We therefore continued the analysis by including both types of variables, i.e. open and total.

	Wilks' Lambda	F	df1	df2	Sig.
Num audiences	,998	3,215	1	1445	,073
Num categories	,991	12,774	1	1445	,000
Num developers	,999	1,308	1	1445	,253
Num licenses	,981	27,754	1	1445	,000
Num pr. languages	,992	12,104	1	1445	,001
Num of runtime dep.	1,000	,702	1	1445	,402
Num open bugs	,998	2,287	1	1445	,131
Num open features	,998	3,585	1	1445	,058
Num open patches	,999	1,756	1	1445	,185
Num open techrequests	,999	1,241	1	1445	,266
Total num bugs	,999	1,531	1	1445	,216
Total num features	,999	2,001	1	1445	,157
Total num techrequests	,999	,892	1	1445	,345
Total num patches	,999	1,487	1	1445	,223
Num forum posts	,999	1,619	1	1445	,203
Tracker activity	,999	1,806	1	1445	,179
Traffic SF logo	1,000	,695	1	1445	,405
Traffic hits	,999	,882	1	1445	,348
Versioning control Read	,998	2,686	1	1445	,101
Versioning control Write	,990	14,170	1	1445	,000
Num downloads	,998	2,460	1	1445	,117
Num releases	,927	113,580	1	1445	,000
Avg time releases	,907	147,597	1	1445	,000
Num newsitems	,989	16,635	1	1445	,000
Num newsposters	,997	3,754	1	1445	,053
Age of project	,969	46,640	1	1445	,000

Table 6.1: Tests of equality of group means

Another important and remarkable determinant is the number of downloads. The F-test as shown in Table 6.1 shows a high Wilk's Lambda for this determinant. Several studies have, however, reported that the number of downloads is an important indicator for being a success or failure (Schweik, 2005; Wang, 2012; Crowston et al., 2006; Mockus et al., 2002; Crowston & Scozzi, 2002;

Crowston et al., 2004; Rai et al., 2002; Midha & Palvia, 2012; Sen et al., 2012). Table 6.1 shows however no significant difference between the two groups (i.e. failed and non-failed).

6.3.1 Box's M

Another assumption when performing MDA is that the variance-co-variance matrices are equivalent. A test to check this assumption is the Box's M test, which tests the null-hypothesis that the covariance matrices do not differ between groups formed by the dependent variable. This test is very sensitive to meeting the assumption of multivariate normality. The assumption of equal dispersion for groups defined by the dependent variable only affects the classification phase of discriminant analysis, and so is not evaluated until we are determining the final accuracy rate of the model. Ideally, we want this test not to be significant, so that the null hypothesis that the groups do not differ can be retained.

Box's M	1885.379
F approx.	89.337
df1	21
df2	4297414.580
Sig.	.000

Table 6.2: Box's M test results

Table 6.2 shows the SPSS output of the Box's M test. We see that this test is in fact significant, implying that we have equal population covariance matrices. Moreover, in case we did violate this assumption, we could still proceed the process. MDA is robust even when the homogeneity of variances assumption is not met, provided the data do not contain important outliers. Moreover, we can request the use of separate group dispersion matrices in the classification phase of the discriminant analysis to see if this improves our accuracy rate. If classification using separate covariance matrices were more accurate by 2% or more, we would report classification accuracy based on this model rather than the one that use within-groups covariance. The accuracy of the model, which will be discussed in subsequent sections, did not improve by using the separate-groups covariance matrix. We continued the analysis by using the within-groups covariance matrix as this test proofed to be not significant.

6.3.2 Eigenvalues

Table 6.3 provides information on the discriminant function. The maximum number of discriminant functions is the number of groups - 1. In our case, we have two groups, failed and non-failed group. This results in a single discriminant function. The eigenvalue indicates the proportion of variance explained, being the between-group sums of squares divided by the within-groups sums of squares. Ideally, we want this number to be large as it is associated with a strong discriminant function. Table 6.3 displays an eigenvalue of .078, implying that the discriminant function is weak.

The canonical correlation is the multiple correlation between the predictors and the discriminant function. This provides an index for the overall model fit, as should be interpreted as the proportion of variance explained (R^2). The canonical correlation of .270 suggests that the model explains approx. 7.3% of the variation in the grouping variable.

Function	Eigenvalue	% variance	cumulative %	Canonical correlation
1	.078	100.0	100.0	.270

Table 6.3: Eigenvalues

6.3.3 Standardized Canonical Coefficients

The standardized discriminant function coefficients (Table 6.4) serve the same purpose as beta weights in multiple regression (partial coefficient): they indicate the relative importance of the independent variables in predicting the dependent. The signs indicate the direction of the relationship. The numbers in the tables allow us to compare variables measured on different scales. Large numbers, when taking the absolute value of a coefficient, correspond to variables with greater discriminant power. Tabel 6.4 furthermore shows the variables that were included in the discriminant function. Determinant *Age of project* has the highest coefficient, being a strong predictor for project failure and non-failure.

	Value
Number of licenses	-0.312
Number of prog. languages	0.293
Number of news posts	0.335
Number of releases	0.313
Age of project	-0.624
Versioning control write	0.309

Table 6.4: Standardized Canonical Discriminant Function Coefficients

6.3.4 Canonical Discriminant Function Coefficients

The unstandardized coefficients are used to create the discriminant function, which can be used in a similar manner as a regression equation. Table 6.5 shows the coefficients with their corresponding value. The discriminant coefficients, the b's, indicate the partial contribution of each variable to the discriminant function controlling for all other variables in the equation.

Translating Table 6.5 results in the following discriminant function:

$$Z = -0.407v_1 + 0.059v_2 + 0.134v_3 + 0.004v_4 - 0.239v_5 + 0.105v_6 + 1.779$$

Where: Z: Discriminant score (Z-score)
v_1: Number of licenses
v_2: Number of programming languages
v_3: Number of news posts
v_4: Number of releases
v_5: Age of project
v_6: Versioning control write

	Value
Number of licenses	-0.407
Number of prog. languages	0.059
Number of news posts	0.134
Number of releases	0.004
Age of project	-0.239
Versioning control write	0.105
Constant	1.779

Table 6.5: Canonical Discriminant Function Coefficients

When calculating the Z-score for open source projects by using the function as displayed above, a higher score implies a higher likelihood to be classified as non-failed. Positive coefficients thus increase the z-score and have a positive effect. On the other hand, negative coefficients causes a lower z-score and hence a higher likelihood that a project is going to be classified as failed. Table 6.5 shows the coefficients with their values. Interestingly, we see that the number of licenses and the age of the project have negative coefficients. This implies that multiple licenses causes the z-score to decrease. Similarly, projects that have been around for some time are more likely to receive a lower z-score as the age of the project has a negative effect on the score. When looking at the age of a project, this effect contradicts a lot of studies on open source success and health, as older projects are more likely to survive and are generally not abandoned. We believe this is caused by the definition of *project failure*. For example, projects that are mature and in existence for years could still be useful and be downloaded frequently, however, project administrators could not produce any more releases as the software is mature enough. According to the definition proposed by English and Schweik (2007), producing no releases in a 12-month period is accompanied with project failure.

6.3.5 Classification Results

Table 6.6 shows the classification table, also known as a confusion matrix. It shows the observed counts (rows) in relation to the predicted counts (columns). Both the original classification, as well as the cross-validated counts are displayed. Cross validation is based on the *leave one out* method. It creates a

function with n-1 cases, and use the function to classify the case that was left out. The process is repeated with each case left out in turn. By performing cross-validation, the analysis produces a more reliable function.

Looking at the cross-validation matrix (Table 6.6), we see that the projects that were labeled as becoming *failed* cannot be predicted with high accuracy. Out of 517 projects, only 18.8% were classified correctly, making errors in 81.2% of the cases. In contrast, projects that were labeled as becoming *non-failed*, i.e. being still in existence with no abandonment and sufficient activity, are classified with high accuracy. Out of 930 cases, the discriminant function is able to classify 90.6% correctly.

The overall classification results show that 65.0% of the cases were classified correctly into failed and non-failed projects when performing cross-validation (model predicts 65.2% correctly without cross-validation). This number is also referred to as the *hit-ratio*. Ideally we want this number to be as high as possible, as it indicates how well we can predict project failure and non-failure.

			Predicted group membership		
		classification	failed	non-failed	total
original	count	failed	99	418	517
		non-failed	85	845	930
	%	failed	19.1	80.9	100
		non-failed	9.1	90.9	100
cross validated	count	failed	97	420	517
		non-failed	87	843	930
	%	failed	18.8	81.2	100
		non-failed	9.4	90.6	100

Table 6.6: Classification results

Prior Probabilities

The classification results as shown in Table 6.6 are calculated based on the prior probabilities of their respective group size. More specifically, we have 517 cases in the failed group and 930 cases in the non-failed group. Without any knowledge, our best guess would be to classify any new project as non-failed, as we are right in 64.3% of the cases (35.7% for the failed group).

We could however change the prior probabilities to 50% for both groups, implying we have no prior knowledge. This would alter our classification results (the discriminant function stays the same). It turns out when doing so, the overall classification power of the model decreases from 65.0% to 61.4%, as it decreases the likelihood that a project is classified as non-failed, which is the largest group (we had 64.3% as prior probability and changed this to 50%). As a result, a greater percentage of the non-failed group is misclassified, and a larger percentage of failed projects is classified correctly (see Table 6.7 for exact percentages).

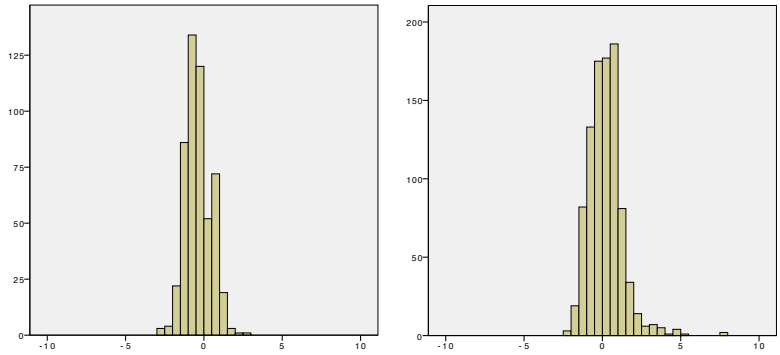
In terms of predicting open source project failure, we increased our classification accuracy from 18.8% to 69.4%. Making the Z-score for open source projects more suitable for predicting project failure when classifying any new project and having no knowledge on prior probabilities, thus using 50% probability for both groups.

		classification	Predicted group membership		
			failed	non-failed	total
original	count	failed	359	158	517
		non-failed	399	531	930
	%	failed	69.4	30.6	100
		non-failed	42.9	57.1	100
cross validated	count	failed	359	158	517
		non-failed	400	530	930
	%	failed	69.4	30.6	100
		non-failed	43.0	57.0	100

Table 6.7: Classification results with equal prior probabilities

6.3.6 Histograms

Figures 6.1(a) and 6.1(b) graphically depict the distribution of the canonical function (discriminant function). It is an alternative way of illustrating the distribution of the discriminant function scores of each group. We see that both distributions have some sort of overlap, implying that the function does not discriminate too well.



(a) Failed group: Mean = -0.38, Std. dev. = 0.814, N = 517

(b) Non-failed group: Mean = 0.21, Std. dev. = 1.090, N = 930

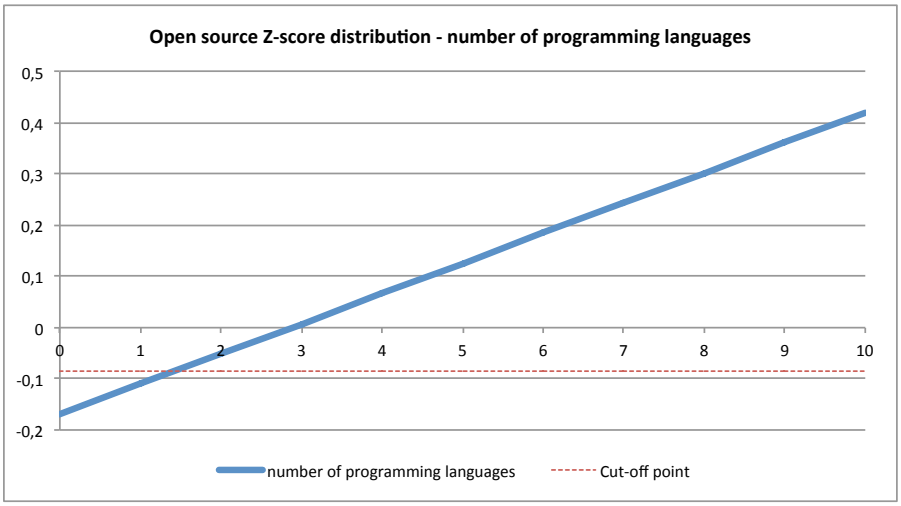
Figure 6.1: Histograms showing the distribution of discriminant scores for failed and non-failed FLOSS projects

6.3.7 Cut-off Point

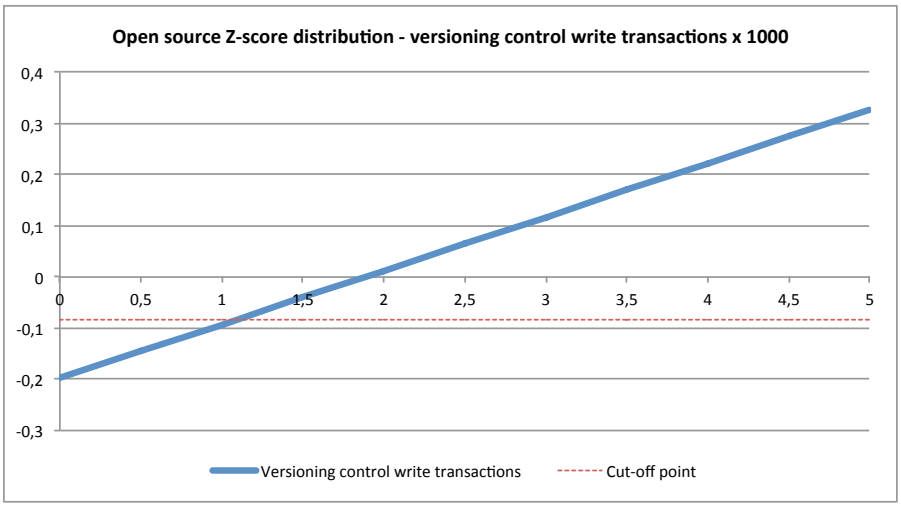
The discriminating function as proposed in Section 6.3.4 is a linear function that takes into account 6 determinants to achieve an overall score, the discriminant score. To classify new projects into failed and non-failed projects, we need to set a threshold, also known as a cut-off point. Values below and above this cut-off point are labeled according to the *a priori* groups that we defined. Looking at the histograms (see Figures 6.1(a) and 6.1(b)), the failed group has a mean score of -0.38 and the non-failed 0.21. The cut-off point is calculated as ((-0.38 + 0.21) / 2) = -0.085. Scores below this value are projects that are classified as failed, values equal or above are destined to be non-failed.

Figures 6.2(a) and 6.2(b) show the linear distribution of the Z-score for FLOSS projects. The predictor variable *number of programming languages* is varied in Figure 6.2(a). Other predictor variables are set to their mean values

(see Table A.1). We see that we need at least two programming languages, together with all other mean values, to be classified as non-failed. Figure 6.2(b) shows a similar graph, but for the number of write transactions in the versioning control system. Note that this number needs to be multiplied by 1000. Approximately 1000 transaction, and other predictors variables with their mean scores, are needed to be classified as non-failed.



(a)



(b)

Figure 6.2: Linear distribution of FLOSS Z-score with predictors (a) *number of programming languages* and (b)*versioning control write transaction*

Chapter 7

Conclusion & Discussion

We have proposed a classification model by performing multiple discriminant analysis on a sample that consisted out of 1,447 open source projects. These projects were labeled as failed (517 cases) or non-failed (930 cases) if in subsequent year they adhere to the definition of failed and non-failed projects. The project failure definition was adopted from English and Schweik (2007). We used an initial set of 25 determinants (variables) that were collected from Sourceforge's research database, SourceForge's APIs and the project pages of SourceForge itself. These determinants were derived from approximately 50 scientific papers on open source health, success and failure. Doing so, we were able to find a set of variables that are essential for the prediction of open source project failure and non-failure. At the end, the model uses 6 variables that best predict the classification of failed and non-failed groups. As our approach of finding such classification model is similar to the construction of the Z-score model for firms (classifying distressed and non-distressed firms), as developed by Altman (Altman, 1968), we named our model: *A Z-score for Open Source Projects*. We used the same approach as Altman but in the context of open source.

The model we constructed was able to classify 65% of the cases correctly by performing cross validation and taking prior probabilities into account (the percentage of cases in a certain group). Setting the prior probabilities to 50%, meaning we have no additional knowledge on group sizes, the model is able to classify 61.4% correctly when using cross validation as well. Although we violated the normality assumption for multiple discriminant analysis, this was not severe as discriminant analysis is quite robust and the significance tests are still thrustworthy. Apart from the overall classification, when we want to predict open source project failure, we can classify approximately 70% of the cases correctly when setting the prior probabilities to 50% (this is a better estimation because for future classification we cannot base our prior probabilities on the group sizes). For example, when taking any new open source project, we can predict project failure with 70% accuracy. This makes the Z-score for open source projects more suitable for predicting open source project failure than non-failure (57.0% of the non-failed cases can be correctly classified).

Due to the complexity of some determinants, and due to the availability of data, we were unable to include some important variables in our study. For example, the number of code commits (Schweik, 2005; Midha & Palvia, 2012;

Samoladas et al., 2010; Rainer & Gale, 2005), Changes LOC in CVS (Schweik, 2005), Skills of developers (Wang, 2012; Crowston et al., 2006) and number of external networks (Wang, 2012; Grewal et al., 2006; Wu et al., 2007) are all essential determinants that are proven to have an important impact on project success. Another important aspect is the construction of the sample. As we were focused on retrieving data on as much determinants as possible, we could have extracted projects that were more mature and stable in their existence. For example, we joined several determinants with an inner join because we were trying to avoid *null* values in our dataset. This reduced the size of the sample considerably, but at the same time provided information on a large selection of determinants. This choice in sample construction could favor more mature projects. On the other hand, we excluded for some determinants empty values. Extracting these cases would result in *null* and 0 values. This could then have two meanings: (1) no data available or (2) 0 activity in that respect.

An important indicator for project success, or the lack of success, is the number of downloads. Several studies have indicated that this determinant reflects project use and that in a large part of the cases it has influenced FLOSS project success (Schweik, 2005; Wang, 2012; Crowston et al., 2006; Mockus et al., 2002; Crowston & Scozzi, 2002; Crowston et al., 2004; Rai et al., 2002; Midha & Palvia, 2012; Sen et al., 2012). When performing multiple discriminant analysis, this determinant seemed to have no significant effect on classifying projects into failed and non-failed, nor has it been included in the stepwise analysis with its relationship to other variables (MDA has the advantage of considering all characteristics of the observation, as well as the interactions between them).

We adopted the open source project failed definition from English and Schweik (2007) and have taken into account their recommendations for improving the definition. During their data collection phase they were unable to retrieve certain determinants which caused their definition to be scoped down. An important remark in this respect, also noted by Wiggins and Crowston (2010), is that a success remains a success, even if a project becomes inactive. This is a conservative classification choice, reflecting the reality that successful projects may enter a retirement stage after active development stops, but in which it is still useful to others. This is another view than English and Schweik's, where a successful project can be abandoned and fall into disrepair. We could therefore argue that projects that do not produce any releases, but are still being downloaded and are found to be useful, should not really be labeled as failed.

Any future work in this respect should try to focus on these remarks. At the end, the classification of projects into dummy variables (i.e. 0 for failed and 1 for non-failed) to perform regression or classification techniques is extremely important. Not only when using multiple discriminant analysis, but also when the researcher would like to try other classification/regression techniques such as logistic regression, classification trees, cluster analysis or machine learning. Furthermore, one should carefully look at the data as this often is a reflection of the project administrator. For example, one might only upload new releases when a significant change has been implemented, others might fade away from what is considered a minor or major update. It could be the case that 5 releases for one project are still less than one release from another. Moreover, certain determinants are not representative indicators of reality. For example, the number of downloads are only counted when the software is directly down-

loaded from the repository, any automatically updated software package are not counted. How does this reflects reality is a point of worry. At the end, open source is diverse, not completely understood and extremely hard to capture in numeric values. Creating regression or classification models are therefore easily impaired by using values that do not really reflect reality. Still, based on the model we created, we were able to predict 70% of the failed projects correctly, which makes it better than flipping a coin.

References

Altman, E. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *Journal of Finance*, 189-209.

Altman, E., Haldeman, R., & Narayanan, P. (1977). Zeta analysis: A new model to identify bankruptcy risk of corporations. *Journal of Banking and Finance*.

Beaver, W. (1967). Financial ratios as predictors of failures. *Journal of Accounting Research*.

Box, G., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society*, *26*(2), 211–252.

Comino, S., Manenti, F., & Parisi, M. (2007). From planning to mature: On the success of open source projects. *Research Policy*, *36*, 1575-1586.

Crowston, K., Annabi, H., & Howison, J. (2003). Defining open source software project success. In *International conference on information systems.*

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Effective work practices for software engineering: Free/libre open source software development. In *Proceedings of the 2004 acm workshop on interdisciplinary software engineering research* (p. 18-26).

Crowston, K., Howison, J., & Annabi, H. (2006). Information systems success in free and open source software development: Theory and measures. *Software Process Improvement and Practice*, *11*, 123-148.

Crowston, K., & Scozzi, B. (2002). Open source software projects as virtual organizations: Competency rallying for software development. In *Iee proceedings software* (Vol. 149, p. 3-17).

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/libre opensource software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, *44*(2), 7.

DeLone, W., & McLean, E. (1992). Information systems success: The quest for the dependent variable. *Information Systems Research*, *3*(1), 60-95.

DeLone, W., & McLean, E. (2003). The delone and mclean model of information system success: A ten-year update. *Journal of Management Information Systems*, *19*(4), 9-30.

English, R., & Schweik, C. (2007). Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects. In *Proceedings of the 1st international workshop on emerging trends in floss research and development.* Minneapolis, MN.

Evangelopoulus, N., Sidorova, A., Fotopoulos, S., & Chengalur-Smith, I. (2009). Determining process death based on censored activity data. *Communications in Statisics, Simulation and Computation*, *37*, 1647-1662.

Feller, J., Fitgerald, B., Hissam, S., & Lakhani, K. (2005). *Perspectives on free and open source software.* Cambridge, MA: MIT Press.

Field, A. (2009). *Discovering statistics using spss* (3rd ed.). London: Sage.

Garett, H. (1968). The tragedy of the commons. *Science*, *162*, 1243–1248.

Grewal, R., Lilien, G., & Mallapragada, G. (2006). Location, location, location: How network embeddedness affects project success in open source systems. *Management Science*, *52*, 1043-1056.

Hann, I., Roberts, J., & Slaughter, A. (2004). Why developers participate in open source software projects: An emperical investigation. In *Proceedings of the 25th international conference on information systems* (p. 821-830).

Kuwabara, K. (2000, March). Linux: A bazaar at the edge of chaos. *First Monday*, *5*(3).

Lee, G., & Cole, R. E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the linux kernel development. *Organization Science*, *14*(6), 633-649.

Lee, S.-Y., Kim, H.-W., & Gupta, S. (2009). Measuring open source software success. *Omega*, *37*, 426-438.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*, *52*.

Lerner, J., & Tirole, J. (2005). The scope of open source licensing. *Journal of Law*, *21*, 20-56.

Levy, Y., & Ellis, T. J. (2006). A systems approach to conduct an effective literature review in support of information systems research. *Informing Science: International Journal of an Emerging Transdiscipline*.

Midha, V., & Palvia, P. (2012). Factors affecting the success of open source software. *The Journal of Systems and Software*, *85*, 895-905.

Mockus, A., Fielding, R., & Herbsleb, J. (2002). Two case studies of open source development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, *11*, 309-346.

OMG. (2004). *Uml 2.0 superstructure specification* (Technical Report ptc/04-10-02). Object Management Group.

Rai, A., Lang, S., & Welker, R. (2002). Assessing the validity of is success models: an empirical and theoretical analysis. *Information Systems Research*, *13*(1), 50-69.

Rainer, A., & Gale, S. (2005). Evaluating the quality and quantity of data on open source software projects. In M. Scotto & G. Succi (Eds.), *Proceedings of the first international conference on open source systems* (p. 29-36). Genova.

Raymond, E. S. (1999). *The cathedral and the bazaar: Musings on linux and open source by an accidental revolutionary.* Sebastopol, California: O'Reilly and Associates.

Samoladas, I., Angelis, L., & Stamelos, I. (2010). Survival analysis on the duration of open source projects. *Information and Software Technology*, *52*, 902-922.

Schweik, C. (2005). An institutional analysis approach to studying libre software 'commons'. *Upgrade: The European Journal for the Informatics Professional*, 17-27.

Schweik, C., & Semenov, A. (2003). The institutional design of open source programming: Implications for addressing complex public policy and management problems. *First Monday*, *8*(1).

Sen, R., Singh, S., & Borle, S. (2012). Open source software success: Measures and analysis. *Decision Support Systems*, *52*, 364-372.

Stallman, R. M. (2010). *Free software, free society: Selected essays of richard m. stallman.* Boston: GNU Press.

Stamelos, I. (2009). Teaching software engineering with free/libre open source projects. *International Journal of Open Source Software and Processes*, *1*, 72-90.

Stewart, K., Ammeter, A., & Maruping, L. (2006). Impacts of license choice and organizational sponsorship on user interest and development activity in open source software products. *Information Systems Research*, *17*, 126-144.

Subramaniam, C., Sen, R., & Nelson, M. (2009). Determinants of open source software projects success: A longitudinal study. *Decision Support Systems*, *46*, 576-585.

Wang, J. (2012). Survival factors for free open source software projects: a multi-stage perspective. *European Management Journal*, *30*, 352-371.

Webster, J., & Watson, R. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, *26*(2), 13-23.

Weerd v.d., I., & Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. *Handbook of research on modern systems analysis and design technologies and applications*, *35*.

Wiggins, A., & Crowston, K. (2010). Reclassifying success tragedy floss projects. *Open Source Software: New Horizons*, 294-307.

Wu, J., Goh, K., & Tang, Q. (2007). Investigating success of open source software projects: A social network perspective. In *Proceedings of international conference on information systems.*

Wynn, D. (2003). Organizational structure of open source projects: A life cycle approach. In *Proceedings of the 7th annual conference of the southern association for information systems* (p. 285-290). Georgia, USA.

Appendix A

Statistical output

	N	Minimum	Maximum	Mean		Std. Deviation	Variance
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic	Statistic
Age of project	1447	,17	12,17	6,4621	,06965	2,64940	7,019
Donate option	1447	0	1	,29	,012	,454	,206
Num downloads	1447	1	8680674	49336,25	10024,871	381340,601	1,454E+11
Forums posts	1447	0	18812	38,50	13,736	522,523	273030,341
Num of audiences	1447	1	10	2,55	,041	1,556	2,420
Num of categories	1447	1	10	2,75	,039	1,467	2,152
Num of developers	1447	1	136	4,96	,258	9,822	96,469
Num of licenses	1447	1	7	1,67	,020	,774	,599
Num news posts	1447	0	44	,80	,066	2,510	6,299
Num news posters	1447	0	3	,27	,012	,461	,213
Number of releases	1447	1	1991	20,56	2,157	82,059	6733,685
Num open bugs	1447	0	5517	7,02	3,928	149,431	22329,667
Num open features	1447	0	101	1,18	,133	5,068	25,680
Num open patches	1447	0	4418	3,34	3,053	116,147	13490,166
Num open tech req.	1447	0	53	,31	,062	2,340	5,474
Num of pr. languages	1447	1	48	2,99	,132	5,009	25,087
Num of runtime dep.	1447	1	8	1,83	,029	1,115	1,244
Versioning control read	1447	1	848217	8806,07	1181,894	44958,607	2,021E+9
Versioning control write	1447	0	44312	822,00	77,912	2963,722	8783647,95
Total num bugs	1447	0	5517	11,56	3,993	151,874	23065,622
Total num features	1447	0	145	1,88	,208	7,915	62,653
Total num patches	1447	0	5125	4,42	3,547	134,915	18201,960
Total num tech req.	1447	0	565	1,27	,460	17,483	305,661
Tracker activity	1447	0	6182	34,17	6,552	249,237	62119,203
Traffic hits	1447	0	26179968	314687,28	37265,996	1417578,11	2,010E+12
Traffic SF logo	1447	0	74003850	156901,49	54319,664	2066290,33	4,270E+12
Valid N (listwise)	1447						

Table A.1: Descriptives of determinants for total sample

	N	Minimum	Maximum	Mean		Std. Deviation
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic
Age of project	517	,17	12,16	7,0902	,11230	2,55341
Donate option	517	0	1	,26	,019	,442
Num downloads	517	1	2708159	28259,60	7132,088	162166,798
Forums posts	517	0	1079	15,07	3,713	84,416
Num of audiences	517	1	9	2,65	,071	1,624
Num of categories	517	1	9	2,94	,068	1,535
Num of developers	517	1	136	5,36	,510	11,606
Num of licenses	517	1	7	1,82	,037	,844
Num news posts	517	0	7	,44	,046	1,045
Num news posters	517	0	3	,24	,019	,443
Number of releases	517	1	265	9,57	,839	19,072
Num open bugs	517	0	5517	14,98	10,979	249,641
Num open features	517	0	48	,84	,138	3,140
Num open patches	517	0	4418	8,76	8,545	194,297
Num open tech req.	517	0	35	,22	,084	1,909
Num of pr. languages	517	1	46	2,38	,165	3,760
Num of runtime dep.	517	1	7	1,86	,048	1,095
Versioning control read	517	1	645840	6209,69	1507,114	34268,203
Versioning control write	517	0	34335	430,42	96,610	2196,674
Total num bugs	517	0	5517	18,18	11,050	251,244
Total num features	517	0	145	1,48	,339	7,716
Total num patches	517	0	5125	10,22	9,913	225,388
Total num tech req.	517	0	208	,69	,411	9,353
Tracker activity	517	0	6182	45,97	17,457	396,930
Traffic hits	517	0	26179968	267733,52	63911,068	1453186,48
Traffic SF logo	517	0	74003850	217644,13	144881,658	3294266,12
Valid N (listwise)	517					

Table A.2: Descriptives of determinants for failed group

	N	Minimum	Maximum	Mean		Std. Deviation	Variance
	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic	Statistic
Age of project	930	,40	12,17	6,1129	,08653	2,63868	6,963
Donate option	930	0	1	,30	,015	,460	,212
Num downloads	930	5	8680674	61053,05	15075,349	459736,344	2,114E+11
Forums posts	930	0	18812	51,53	21,265	648,492	420541,509
Num of audiences	930	1	10	2,50	,050	1,514	2,293
Num of categories	930	1	10	2,65	,047	1,418	2,012
Num of developers	930	1	132	4,74	,284	8,672	75,198
Num of licenses	930	1	7	1,59	,024	,720	,519
Num news posts	930	0	44	1,00	,099	3,014	9,087
Num news posters	930	0	3	,28	,015	,470	,221
Number of releases	930	1	1991	26,68	3,308	100,868	10174,379
Num open bugs	930	0	107	2,59	,305	9,300	86,489
Num open features	930	0	101	1,36	,192	5,865	34,397
Num open patches	930	0	34	,32	,063	1,935	3,745
Num open tech req.	930	0	53	,36	,084	2,547	6,488
Num of pr. languages	930	1	48	3,33	,182	5,556	30,870
Num of runtime dep.	930	1	8	1,81	,037	1,126	1,269
Versioning control read	930	1	848217	10249,43	1635,642	49880,364	2,488E+9
Versioning control write	930	0	44312	1039,68	108,057	3295,285	10858900,2
Total num bugs	930	0	382	7,87	,929	28,333	802,770
Total num features	930	0	119	2,10	,263	8,020	64,319
Total num patches	930	0	162	1,19	,305	9,298	86,448
Total num tech req.	930	0	565	1,59	,678	20,661	426,883
Tracker activity	930	0	1172	27,60	3,121	95,172	9057,798
Traffic hits	930	180	23336885	340789,53	45825,796	1397498,97	1,953E+12
Traffic SF logo	930	0	16905766	123133,81	25709,825	784044,281	6,147E+11
Valid N (listwise)	930						

Table A.3: Descriptives of determinants for non-failed group

		Num audiences	Num categories	Num developers	Num licenses	Num pr. languages	Num of runtime dep.	Num open bugs	Num open features	Num open patches	Num open tech requests	Total num bugs	Total num features	Total num techrequests	Total num patches	Num forum posts	Tracker activity	Traffic SF logo	Traffic hits	Versioning control Read	Versioning control Write	Num downloads	Num releases	Avg time releases	Num newsitems	Num newsposters	Age of project
Correlation	Num audiences	1,00	,494	,103	,279	,060	,232	-,030	,026	-,01	,060	-,02	,047	,031	-,01	,017	-,01	,013	,036	,063	,127	,036	,022	,004	,032	,046	,244
	Num categories	,494	1,000	,142	,378	,108	,314	-,002	,028	,038	,045	,007	,031	,038	,039	,009	,036	,058	,064	,090	,111	,061	,038	,015	,081	,081	,277
	Num developers	,103	,142	1,000	,163	,248	,207	,053	,154	,00	,042	,082	,147	,015	,010	,034	,123	,207	,267	,399	,248	,285	,177	,017	,034	,054	,261
	Num licenses	,279	,378	,163	1,00	,039	,257	,003	,005	,006	,018	,024	-,01	,019	,006	-,01	,028	,013	,035	,076	,030	,028	,017	,005	,023	-,05	,337
	Num pr. languages	,060	,108	,248	,039	1,00	,098	,007	,170	-,01	,135	,029	,131	,028	,00	,018	,059	,263	,169	,144	,145	,231	,047	-,01	,090	,097	,098
	Num of runtime dep.	,232	,314	,207	,257	,098	1,00	,008	,052	-,02	,037	,020	,049	,064	-,01	-,02	,029	,117	,098	,123	,149	,093	,122	-,01	,094	,025	,280
	Num open bugs	-,03	-,002	,053	,003	,007	,008	1,000	,031	,00	,011	,988	,024	,059	,036	,002	,738	,00	,026	,013	,009	,006	,015	,106	-,004	-,01	,005
	Num open features	,026	,026	,154	,005	,170	,052	,031	1,00	,027	,194	,084	,846	,059	,076	,002	,178	,021	,357	,184	,159	,086	,182	-,01	,056	,080	,069
	Num open patches	-,01	,038	-,003	,006	-,01	-,02	-,001	,027	1,00	,010	,00	,016	,001	,999	,002	,535	,00	,010	,002	,00	,001	-,01	-,02	-,005	-,01	-,010
	Num open techrequests	,060	,045	,042	,018	,135	,037	,011	,194	,010	1,00	,032	,174	,296	,017	,015	,129	,018	,148	,094	,053	,062	,075	-,01	,087	,049	,065
	Total num bugs	-,02	,007	,082	,024	,029	,020	,988	,084	,00	,032	1,00	,087	,011	,001	,010	,771	,006	,094	,046	,037	,016	,017	,105	,008	,00	,020
	Total num features	,047	,031	,147	,031	,131	,011	,024	,846	,016	,174	,087	1,00	,059	,023	,066	,186	,019	,338	,168	,166	,061	,155	-,01	,075	,081	,067
	Total num techrequests	,031	,038	,015	,019	,028	,064	,002	,059	,001	,296	,011	,059	1,00	,002	,005	,164	,004	,075	,037	,018	,018	,005	-,01	,074	,060	,025
	Total num patches	-,01	,039	,010	,006	,006	-,01	-,001	,036	,999	,017	,001	,023	,002	1,00	,002	,541	,001	,027	,009	,003	,003	-,01	-,02	-,004	-,01	-,004
	Num forum posts	,017	,009	,034	-,01	,018	-,02	,002	,076	,002	,015	,010	,066	,005	,002	1,00	,023	,004	,061	,008	,046	,014	,016	,004	,055	,078	,000
	Tracker activity	-,01	,036	,123	,028	,059	,029	,738	,178	,029	,032	,771	,087	,164	,541	,023	1,00	,013	,188	,104	,076	,031	,014	,152	,028	,007	,043
	Traffic SF logo	,013	,058	,207	,013	,263	,117	-,001	,021	,00	,018	,006	,019	,004	,001	,004	,013	1,00	,103	,056	,008	,247	,030	,007	,033	,056	,087
	Traffic hits	,036	,064	,267	,035	,169	,098	,026	,357	,010	,148	,094	,338	,075	,027	,061	,188	,103	1,00	,313	,111	,150	,027	,00	,086	,106	,166
	Versioning control Read	,063	,090	,399	,076	,144	,123	,013	,184	,002	,094	,046	,168	,037	,009	,008	,104	,056	,313	1,00	,248	,116	,041	,013	,018	,049	,175
	Versioning control Write	,127	,111	,248	,030	,145	,149	,009	,159	,00	,053	,037	,166	,018	,003	,046	,076	,008	,111	,248	1,00	,038	,029	,007	,085	,052	,066
	Num downloads	,036	,061	,285	,028	,231	,093	,006	,086	,001	,062	,016	,061	,018	,003	,014	,031	,247	,150	,116	,038	1,00	,023	,002	,052	,002	,080
	Num releases	,022	,038	,177	,017	,047	,122	,015	,182	-,01	,075	,017	,155	,005	-,01	,016	,014	,030	,027	,041	,029	,023	1,00	-,01	,008	,053	,075
	Avg time releases	,004	,015	,017	,005	-,01	-,01	,106	-,01	-,02	-,01	,105	-,01	-,01	-,02	,004	,152	,007	,00	,013	,007	,002	-,01	1,00	-,005	-,01	,081
	Num newsitems	,032	,081	,034	,023	,090	,094	-,004	,056	,00	,087	,008	,075	,074	,00	,055	,028	,033	,086	,018	,085	-,01	,008	-,01	1,000	,539	,049
	Num newsposters	,046	,081	,054	-,05	,097	,025	-,012	,080	-,01	,049	,00	,081	,060	-,01	,078	,007	,056	,106	,049	,052	-,02	,053	-,01	,539	1,00	,017
	Age of project	,244	,277	,261	,337	,098	,280	,005	,069	-,01	,065	,020	,067	,025	,00	,000	,043	,087	,166	,175	,066	,080	,075	,081	,049	,017	1,000

Table A.4: Pooled within groups matrix