OPEN UNIVERSITY UK

MASTER DISSERTATION

Evaluating phylogenetic methods for quantifying risks and opportunities presented by forks in open source software

Author:
Alvaro ORTIZ TRONCOSO

Supervisor:
Dr. Doug LEITH

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Computing (Software Engineering)

8. September 2017

OPEN UNIVERSITY UK

Abstract

Master of Science in Computing (Software Engineering)

Evaluating phylogenetic methods for quantifying risks and opportunities presented by forks in open source software

by Alvaro ORTIZ TRONCOSO

Software needs to evolve in order to deliver value to its stakeholders throughout its lifecycle. The most important drivers behind software evolution are changing user expectations, and a dynamic and innovative ecosystem. Open source software development is a software development paradigm that embraces change by blurring the distinction between users and developers. The cornerstone of open source development is a licensing scheme that grants anybody the right to examine, to copy and to modify the source code. Open source licenses have proven in many cases a viable alternative to strong intellectual property protection regimes. However, open source licenses expose software projects to a new kind of risk: forking. Forking happens when part of the team takes off in a new direction. Literature on the governance of open source projects disagrees on whether forking is a risk or an opportunity: the traditional view is that forking is the result of a failure of the project to keep its resources together however, as successful software products have lately emerged from forks, the traditional view is challenged. Notwithstanding, methods for quantifying the evolution of forks are currently scarce: the present research attempts to port methods from phylogenetics, a branch of evolutionary biology that attempts to unravel the mechanisms behind the evolution of living organisms, to the study of the evolution of forks and postulates that the progress of a fork can be modelled using these methods. Methods and concepts from evolutionary biology were validated by applying them to three cases of software forks. A statistical analysis shows that the history of a forked project can be reconstructed using phylogenetic trees, and finds evidence that the eventuality of a fork could be predicted. However, no evidence was found that the outcome of a fork can be foretold using these methods. The present research concludes by porting basic concepts from evolutionary biology into a software development context and elaborates how phylogenetic methods and concepts can be used by practitioners to increase their understanding of forking processes.

Acknowledgements

I am extremely grateful to the Open University staff, who have provided me with such professional and well informed support. Special mention must be made of my tutor, Dr. Doug Leith, whose guidance has been invaluable throughout. I also would like to thank my colleagues at the Technical University Berlin, in particular Erhard Zorn and Dr. Stefan Born, for balancing a busy workload. I am especially thankful for the encouragement shown by my family and in particular for the patience afforded me by my dear friend Ina Kemter.

Contents

Abstract i

Acknowledgements ii

Glossary v

1 Introduction 1
- 1.1 Background to the problem . . . 1
- 1.2 Justification for the research . . . 1
- 1.3 Definitions . . . 2
- 1.4 Scope of the research . . . 2
- 1.5 Aim . . . 3
- 1.6 Outline of the dissertation . . . 3

2 Research definition 4
- 2.1 The practical problem . . . 4
- 2.2 Existing relevant knowledge . . . 4
  - 2.2.1 Software metrics . . . 5
  - 2.2.2 Evolution of software systems . . . 5
  - 2.2.3 Relevant evolutionary techniques . . . 6
  - 2.2.4 Software forks as risk or opportunity . . . 6
  - 2.2.5 Application of methods from evolutionary biology to software evolution . . . 7
- 2.3 Objectives . . . 8
- 2.4 Summary of Chapter 2 . . . 8

3 Methodology 9
- 3.1 Methods and techniques selected . . . 9
- 3.2 Data acquisition and encoding techniques . . . 9
  - 3.2.1 Code structure characteristics . . . 9
  - 3.2.2 Organizational characteristics . . . 9
  - 3.2.3 Techniques for computing the distance matrix . . . 10
  - 3.2.4 Estimating the phylogeny of releases . . . 10
  - 3.2.5 Cophenetic distance matrix . . . 11
- 3.3 Justification . . . 11
  - 3.3.1 RQ1 . . . 11
  - 3.3.2 RQ2 . . . 12
  - 3.3.3 Research procedures . . . 12
    - Data acquisition and re-encoding . . . 12
    - Distance matrices . . . 13
    - Phylogenetic trees and cophenetic distances . . . 13
    - Analysis of variance . . . 13
  - 3.3.4 Ethical considerations . . . 14

3.3.5 Summary of Chapter 3 14

Glossary

Branch
A thread of development within a project or team; branches are common in open source development (Robles and González-Barahona, 2012).

Evolution
Defined in paragraph 1.3.

Fork
Defined in paragraph 1.3.

Merge
A rejoining of separate development strands that had branched or forked previously, either by integrating source code or by dismissing parts of either project (Robles and González-Barahona, 2012).

Open source software
Software development paradigm that blurs the difference between users and developers (Hippel and Krogh, 2003). Open source software licenses grant users the right to fork a project (Robles and González-Barahona, 2012).

Phylogenetic tree
A pictorial representation of the degree of relationship between entities sharing a common ancestry (Baum and Offner, 2008).

Release
A stage in the software lifecycle corresponding to a new generation of the system (Lehmann, 1980).

Chapter 1

Introduction

1.1 Background to the problem

A software system is constantly subject to change pressure from its environment, therefore it needs to evolve to deliver value to its stakeholders throughout its lifetime: it has been regarded at least since the 1980's that software evolution is the most expensive part of the software lifecycle (Lehman, 1980). Therefore, an understanding of the capacity of a system to adapt to changes in its environment can impact on the software production process (Yu and Ramaswamy, 2006).

Open source development challenges traditional best practices in software development by blurring the difference between users and developers (Hippel and Krogh, 2003). The open source development model originated in the academic community and has transcended into private enterprise, where in many cases it has proven an efficient alternative to strong intellectual property protection (Kogut and Metiu, 2001).

Common tasks in software evolution include cloning, branching and merging of the code base; in addition to these processes, the open source license grants the freedom to fork a project: open source projects can evolve in parallel, splitting in two or more different projects, steered by different development teams.

The gain in popularity of the open source development model has brought forward the importance of understanding forking processes. Kogut and Metiu (2001) argue that forking results in competing versions of the original project and that forking is therefore a major failure risk for open source projects. Nyman and Lindman (2013) argue that forking is a remedy against ailments of proprietary software (planned obsolescence, vendor lock-in, hostile takeovers, etc.) and that forking facilitates experimentation. Robles and González-Barahona (2012) suggest that a purposeful fork can solve technical-, license- and team-related problems by restoring the balance between the stakeholders of a project. Therefore, a controversy exists within the software engineering community, whether forking is a risk or an opportunity.

1.2 Justification for the research

Concepts in software evolution and biological evolution are often described using a common vocabulary (Yu and Ramaswamy, 2006), so it seems natural to look at evolutionary biology for potential methods to solve this controversy.

Lehman (1980) postulates that there are recurring patterns which govern software evolution, independently of the decisions taken by individual managers and programmers. Therefore, software evolution might, as biological evolution, be understood as a process which is not designed, but resulting from characteristics inherent to a population: A population need not be composed of biological entities, characteristics need not be encoded in DNA and the environment can be artificial,

thus the term "evolution" can be used to describe processes in different domains, as long as the population considered ensures its own perpetuation (Nehaniv et al., 2006). If the term "software system" is taken to encompass a system's infrastructure, code base and community of developers (Yu and Ramaswamy, 2006), then a software system is capable of sustaining itself, and thus the change processes affecting a population of software releases can be described as evolution (figure 1.1).

1.3 Definitions

Based on the work by Robles and González-Barahona (2012) a working definition of a fork can be formulated as follows:

Fork

A fork is a bifurcation from an existing project, resulting in an autonomous development strand, with its own name, infrastructure, code base and community of developers.

Based on the work by Nehaniv et al. (2006) and Yu and Ramaswamy (2006), a working definition of the evolution of software can be formulated as follows:

Software evolution

The evolution of software is a process which affects a population of software releases. Software releases are characterized by code and organizational resources. Software evolution is distinct from the governance of a software project, as software evolution is independent of the decisions taken by individual managers and programmers.

1.4 Scope of the research

Forking is a process explicitly enabled by open source software licenses; forking is contrary to strong intellectual property protection; therefore the scope of the research is limited to open source software. Software forks are known to have occurred in the areas of networking, web applications, development environments, multimedia, games, operating- and desktop- systems, utilities, graphics software, databases, enterprise resource planning, security and package management (Robles and González-Barahona, 2012), thus affecting a large portion of the software development domain.

Some projects that have gained a large user base originated from forks, for example the MariaDB database engine, the Android operating system and the LibreOffice suite of office applications, forked for different reasons and with different outcomes. Rather than trying to gain a comprehensive overview of the impact of forking on software development, a task that was undertaken by Robles and González-Barahona (2012), the present research examined three forks: MySQL / MariaDB, Linux / Android and OpenOffice / LibreOffice. Data was collected from the projects' online repositories, without interacting directly with the developers ("third degree data"). This approach entails that data cannot be controlled nor its quality be assessed through other means, therefore, the availability and completeness of the data archived in the repositories was a factor that played an important role in the choice of projects to examine.

Using real-world case studies for describing a software development situation has been practiced in computer science, and it is possible to empirically test hypotheses using this approach (Runeson and Höst, 2009). Hypotheses were formulated as research questions, presented in paragraph 2.1.

1.5 Aim

Any organization aiming to adopt open source software development might face a fork situation during the software's lifecycle, or might decide to fork existing software as a means to solve technical-, license- and team-related problems or to facilitate experimentation, as suggested by Robles and González-Barahona (2012). The aim of this research is to gain empirical evidence of whether phylogenetic methods from evolutionary biology can quantify the risks and opportunities associated with software forking processes.

To this end, the present research reviewed the current state of literature on forking, selected suitable open source software repositories to collect data and implemented selected phylogenetic methods using appropriate libraries. An attempt was made to advance the understanding of forking processes by answering the research questions detailed in chapter 2.

1.6 Outline of the dissertation

The rest of the dissertation is organized as follows: chapter 2 defines the research questions, reviews existing literature and examines the objectives in detail. Chapter 3 examines the data acquisition process, justifies the choice of phylogenetic techniques and relates the chosen statistical techniques to the research questions. Chapter 4 details how the data was acquired, processed using phylogenetic techniques and analysed using statistical techniques. Chapter 5 concludes, delineates possible further research and reflects on the research process as it was carried out.

Chapter 2

Research definition

2.1 The practical problem

Defining a method to quantify the evolution of forks would mean that forking is no more a step in the dark, but instead could become a powerful tool for solving recurring problems of software development. The practical problem was examined here by answering two research questions:

Research Question 1
Can a threshold be determined beyond which two diverging development branches will be more likely to fork than to merge?

If forking is a risk (Kogut and Metiu, 2001), then knowing whether a fork is about to happen can facilitate the choice of a contingency strategy. A metric of the dissimilarity between parallel development strands could detect whether development is about to cross a threshold beyond which a fork will be unavoidable. The value of this threshold can be approximated by examining the release history of well-known forks.

Research Question 2
Can the likely outcome (cooperation, competition or discontinuation) of an ongoing fork be predicted?

Nyman and Lindman (2013) argue that the right to fork inherent in open source licenses explains the resilience and innovation potential of the open source sector, however this potential can only be released if an ongoing fork is heading in the right direction, i.e. if separate teams are cooperating on a shared code base. Forks with known different outcomes: competition (e.g. MySQL/MariaDB), cooperation (e.g. Linux/Android), discontinuation of a branch (e.g. OpenOffice/LibreOffice) can be examined to answer the second research question.

2.2 Existing relevant knowledge

This literature review is structured as follows: (2.2.1) software metrics, (2.2.2) evolution of software systems, (2.2.3) relevant evolutionary techniques, (2.2.4) software forks as risk or opportunity, (2.2.5) application of methods from evolutionary biology to software evolution.

2.2.1 Software metrics

Nagappan et al. (2008) differentiate between methods for quantifying code structure and methods for quantifying organizational structure and evaluate the applicability of these methods for predicting fault-proneness of individual software components. As forking has technical and organizational aspects, the organizational metrics introduced by the authors are especially relevant. The purpose of Nagappan et al. (2008) is different from the problem at hand; however, their comprehensive overview of software metrics and their rationales can provide guidance on which metrics to choose for data collection.
Nagappan et al. (2008) list the following metrics for quantifying code structure:

- "Code churn" is a measure of all the changes to a code base throughout its history.
- "Code complexity" is primarily relevant to object-oriented software and is obtained from the number of methods and inheritance depth of each class.
- "Code dependencies" measures how much the software depends on external components.
- "Code coverage" examines the percentage of code covered by automated tests.

Nagappan et al. (2008) list the following metrics for quantifying organizational structure:

- "Number of engineers" and "Number of ex-engineers" are measures of how much knowledge remains in the team and how much has been lost.
- "Edit frequency" counts the number of changes contributed by each engineer.
- "Depth of ownership" tries to quantify at which organizational level the decisions concerning the software are being taken.
- "Percentage of organization contributing to development" and "Organizational code ownership" attempt to measure the communication overhead resulting from development inside an organization.
- "Organizational code ownership" measures the diversity of contributors.

2.2.2 Evolution of software systems

An early attempt to describe the software maintenance process as evolution can be credited to Lehman (1980). In this insightful paper, Lehman (1980) postulates that there are recurring patterns which govern software evolution, independently of the decisions taken by individual managers and programmers. These patterns are colloquially known as "Lehman's laws". Particularly the third law: "The Fundamental Law of Program Evolution" is of interest here, as it describes the software process as a self-regulating system, composed not only of the source code, but also of the organizations, managers, programmers and users involved. Through a case study, Lehman (1980) gathers empirical evidence for his hypothesis. This paper pioneers at least three interesting points: software evolves through constant interaction with its environment, the software production process is a self-regulating system, and quantitative analysis can improve project planning.

Nehaniv et al. (2006) concentrate on the differences between biological and software evolution. This paper shows the limits of the "software evolution" metaphor: studies on software evolution are yet to agree on a clear-cut definition of what constitutes a population, an individual, a species or heritable characteristics in a software context. Yu and Ramaswamy (2006) concentrate on evolutionary aspects common to biological systems and software systems and try to delineate solutions for the issues raised by Nehaniv et al. (2006). Yu and Ramaswamy (2006) maintain that if the term "software system" is defined so as to include a system's infrastructure, code base and community of developers, then a software system is capable of sustaining itself and therefore the term evolution is applicable.

2.2.3 Relevant evolutionary techniques

The field of taxonomy is concerned with classifying organisms based on their observable characteristics; by contrast, the field of cladistics is concerned with reconstructing biological lineages (Rohlf, 2013). A third approach, phylogenetics, seeks to unravel the mechanisms responsible for the diversity of lineages (Garamszegi and Gonzalez-Voyer, 2014). These fields have in common that they apply statistical methods to generate hypotheses about the evolutionary relationships of populations. These hypotheses are commonly presented using phylogenetic trees (Baum, Offner, 2008).

Among the methods available for estimating phylogenetic trees, "distance matrix methods", i.e. methods that use a matrix of pairwise dissimilarity between observable characters, have been found to produce accurate results (Felsenstein and Felenstein, 2004) and to perform well on large data sets (Desper and Gascuel, 2005). One of the earliest distance matrix methods, "Unweighted Pair Group Method with Arithmetic Mean" (UPGMA), can be traced back to the work of Sneath and Sokal on numerical taxonomy during the 1960's. In a paper published in the influential magazine Nature, Sneath and Sokal (1962) outline the principles underlying UPGMA: organisms presenting a high proportion of common characters are grouped together using statistics and all characters are considered equal in weight. This constitutes a departure from taxonomical practice hitherto, where classification was based on the presence of particular features selected by a taxonomist. The principal claim of the authors is that the statistical nature of numerical taxonomy guarantees the objectivity and repeatability of the classification.

Distance methods estimate a phylogenetic tree by computing a matrix of pairwise dissimilarity between the entities to be classified, and then applying a classification algorithm to the matrix. However, a matrix can theoretically result in multiple trees, and the choice of algorithm affects the topology of the resulting tree (Paradis, 2011). Using a computer is a prerequisite for applying these techniques and several software libraries help to achieve this goal. Paradis (2011) provides up-to-date guidance on how to store, handle and analyse phylogenetic data using the R-language for statistical computing.

2.2.4 Software forks as risk or opportunity

Kogut and Metiu (2001) maintain that software development under an open source licensing regime is more efficient than under strong intellectual property rights, and that the resulting community-based cooperation is more efficient than in-house competitive development. By examining the development processes in two open source

projects: the Linux kernel and the Apache web server, Kogut and Metiu (2001) conclude that governance structures in open source projects are shaped by the need to ward off the risk of forking. Additionally, the authors relate the often modular structure of open source programs to this very risk. Kogut and Metiu deduce that forking is the major risk faced by open source development, and that forking must be avoided as otherwise the community of developers would fragment and the project would lose the competitive edge it gained through open source licensing. This paper provides evidence for considering forks as risks.

Nyman and Lindman (2013) adopt the opposite view. They argue that forking is not intrinsically negative, a synonym for wasted resources, incompatible versions and a risk to the continuation of a project. On the contrary, they see the liberty to fork, which is inherent in open source licenses, as a remedy to recurring problems of proprietary software: planned obsolescence, "hijacked" projects etc. Additionally, Nyman and Lindman (2013) discuss business models built around forks. These are able to thrive in what the authors call a software ecosystem, and are facilitated by a thorough knowledge of the possibilities of open source licenses. Nyman and Lindman's paper does not provide in-depth case studies or a systematic review. The authors make a contribution to the study of forks nonetheless, by placing the practice of forking within a wider technological, managerial and economical context. This paper sets the background for considering forks as opportunities.

Robles and González-Barahona (2012) conduct a comprehensive study of software forks. This paper provides a definition of forking and statistics regarding the relative frequency of forks, their reasons and outcomes.

2.2.5 Application of methods from evolutionary biology to software evolution

A search for the keywords "software fork", "cladistics", "phylogenetic" and "taxonomy" in the Open University library, the Basel Academic Search Engine and Google Scholar yielded no relevant articles, therefore to my knowledge, no study to date links evolutionary techniques to software forking. However, several researchers have studied the application of methods from evolutionary biology to different areas of software evolution.

Tenev and Duszynski (2012) study the evolution of six open source operating systems of the BSD family. Tenev and Duszynski (2012) use algorithms from biological genetics to study the similarity between lines of source code. They propose a new algorithm that can detect similarities of source code, even when files have been renamed or moved. Their algorithm is based on techniques used in genetics; however, they do not present a clear-cut solution to the problem of what constitutes a gene in a software context, a problem raised by Nehaniv et al. (2006). Additionally, their study is limited to characteristics of the code base, and they do not consider any team characteristics.

Nevertheless, Tenev and Duszynski (2012) are able to construct relatively simple evolutionary trees and claim that these trees represent the correct evolution of the BSD family, although it remains unclear how this claim can be substantiated.

Benlarabi et al. (2015) attempt a 'cladistic' approach to the study of variants in a software product line. Software variants are produced by cloning and customizing existing software, within the limits of a common platform. Their study aims at predicting future changes and user requirements. Benlarabi et al. (2015) use the functional features of each software variant as characters. They encode the presence or absence of each feature in a matrix and construct a phylogenetic tree by using

clustering. Benlarabi et al. (2015) use three measures to analyse the trees: the number of shared features, the distance between variants and the number of derived products from each variant. Although the choice of characters used by Benlarabi et al. (2015) is an interesting approach, they limit themselves to using a general hierarchical clustering algorithm and therefore do not exploit the full potential of phylogenetic methods.

2.3 Objectives

As specified in paragraph 1.5, the aim of this research was to gain empirical evidence of whether phylogenetic methods from evolutionary biology can quantify the risks and opportunities associated with software forking processes. It was sought to achieve this aim by answering the two research questions specified in paragraph 2.1. It was attempted to answer these research questions by pursuing the objectives shown in the goal diagram in figure 2.1 and described below.

2.4 Summary of Chapter 2

The concept of evolution has been used at least since 1980 in a software development context. Moreover, there is an abundant body of work on statistical methods for quantifying and describing biological evolutionary processes. However, although quantitative methods have been used to facilitate project management in different use cases, such as predicting fault-proneness of software, no work was found which applies biological evolutionary techniques to quantify the evolution of software forks. Therefore, primary research was required to answer the research questions specified in paragraph 2.1. In order to carry out this research, 5 objectives were defined: reviewing the current state of research, selecting suitable open source repositories to collect data about forks, porting methods from evolutionary biology to analyse the data acquired from the repositories, and reporting the results for an audience of practitioners.

Chapter 3

Methodology

3.1 Methods and techniques selected

In the following sections the term "technique" is used for any procedure that can be boiled down to an algorithm, while the term "method" is used for a coherent combination of techniques.

As illustrated in figure 1.1, and further elaborated in the working definition of "software evolution" in paragraph 1.3, the methods and techniques chosen are applied to a population of software releases. Therefore, the basic entity under scrutiny, the "operational taxonomic unit" (Sokal, 1983), is the "software release".

3.2 Data acquisition and encoding techniques

The choice of techniques for data acquisition is based, with alterations, on the overview of software metrics provided by Nagappan et al. (2008) reviewed in paragraph 2.2.1. Nagappan et al. (2008) distinguish between code structure characteristics and organizational characteristics. The following characteristics were used to describe software releases.

3.2.1 Code structure characteristics

For the present research, it was assumed that for the study of software evolution, the most relevant code structure characteristic is a measure of code churn, redefined as the amount of change introduced by each release.
Code churn was recorded by 2 measurements:

- The presence (or absence) of each file in each specific release.
- a checksum of each file indicating file changes across releases.

3.2.2 Organizational characteristics

Software source code is generally developed and stored in a code repository. Code repositories record who edited each file and who committed the change in the repository, thus providing information about the structure of the development team involved in each software release. Team structure characteristics were recorded in three measurements:

- *Team composition* was recorded as the presence (or absence) of a team member for each specific release.

- *Edit frequency* was recorded as the number of edits by each team member for each release. Editors contribute their knowledge to the code base, so the number of edits indicates how much knowledge is provided by a team.
- *Code ownership* was recorded as the number of commits by each team member for each release. Committers control which edits make it into the code base, so the number of commits indicates how much a release is owned by a team.

3.2.3 Techniques for computing the distance matrix

As discussed in the review of literature on evolutionary techniques (2.2.3), "distance matrix methods", i.e. evolutionary techniques that use a matrix of pairwise dissimilarity between entities to estimate a classification (Felsenstein and Felenstein, 2004) offer an approach that produces accurate results and performs well on large data sets.

The distance matrix is computed on the base of the measurements obtained using the techniques described in paragraph 3.1.1. The result is a matrix with one column and one row for each software release, were each cell contains a measure of the pairwise dissimilarity between the corresponding two releases. As the measurements used are of different data types (table 3.1), a technique that can calculate the distance matrix by combining measurements of mixed data types is required. The Gower distance provided by the R-package "StatMatch" provides such an algorithm (D'Orazio, 2016).

3.2.4 Estimating the phylogeny of releases

Having computed a distance matrix using the techniques described in paragraph 3.1.2, the next step is to estimate a phylogenetic tree that represents the relationships between the software releases studied. The choice of techniques for estimating trees is based on the reference provided by Paradis (2011), as discussed in 2.2.2. Commonly used techniques for estimating trees include:

Average linkage (UPGMA)
: UPGMA is a method that generates a tree based on the degree of similarity between entities (Sneath and Sokal, 1962), but does not imply a descent from a common ancestor. Therefore it is a "phenetic" method, i.e. the result is a hierarchical clustering of the entities under scrutiny, based on their characteristics (Rohlf, 2013).

Neighbour-Joining (NJ)
: NJ is a "cladistic" method, i.e. it assumes a descent from common ancestry and seeks the tree with the fewest possible evolutionary changes, i.e. the tree with the "maximum parsimony" (Rohlf, 2013). As working on trees is a complex problem, a heuristic is used to make the search more efficient: NJ seeks the tree with the shortest total branch length, i.e. the tree with the least amount of changes, and accomplishes this by iteratively re-computing the dissimilarity matrix (Saitou and Nei, 1987).

Minimum Evolution (ME)
: ME is similar to NJ in its aims, but it seeks the shortest tree by applying a different heuristic: seeking the tree with the shortest overall branch length by rearranging the topology of the tree (Desper and Gascuel, 2005).

3.2.5 Cophenetic distance matrix

The cophenetic distance is the distance at which two tips of a tree combine into a single branch (R Core Team, 2017, p. 1275). A cophenetic distance matrix can be calculated for the pairwise distances as represented in the tree. Cophenetic distance matrices were used here for two purposes:

- **As a measure of accuracy**
  The correlation between the distance matrix (obtained from measurements) and the cophenetic distance matrix (obtained from the tree) is a way of telling how well the tree represents the data and can be used to assess the accuracy of the phylogenetic methods (Sokal, 1986).
- **As input for further analysis**
  As the cophenetic distance matrix represents the tree, it can be used as input for statistical analysis. Cophenetic matrices were used to answer the research questions, as detailed in the following paragraph.

3.3 Justification

The practical problem (2.1) was tackled through two research questions. These research questions were answered by reformulating them as statistical hypotheses based on the trees obtained by applying the methods and techniques selected. Cophenetic distance matrices (3.1.4) were instrumental in defining these statistical hypotheses, as cophenetic distance matrices represent the trees in a way that can be used as input for statistical analysis.

3.3.1 RQ1

Research Question 1 sought to determine a threshold beyond which two diverging branches will be more likely to fork than to merge. This result was expected to provide guidance as to the preferred strategy to follow when part of the team is getting off on its own by answering the question: is a merge still practicable or is a fork unavoidable?

Null hypothesis
The null hypothesis is that branches and forks are indistinguishable by measuring the pairwise distance between releases.

Alternative hypothesis
The alternative hypothesis is that a branch can be told from a fork by measuring the pairwise distance between releases.

These hypotheses can be translated into statistical hypotheses by examining the cophenetic distance matrix obtained from the estimated phylogenetic tree.

Statistical null hypothesis
The statistical null hypothesis is that the mean cophenetic distances between releases on the same branch and on different forks are not significantly different.

Alternative statistical hypothesis
The alternative statistical hypothesis is that the mean cophenetic distance between a pair of releases is significantly different ($p < 0.05$) depending on whether the releases are on different forks or on the same branch.

3.3.2 RQ2

Research Question 2 sought to determine whether the outcome of a fork can be predicted through phylogenetic methods. The predicted outcome was expected to fall into one of three categories: cooperation, competition or discontinuation. This result would provide guidance as to which outcome is to be expected in the future of the fork and which strategy is more suitable to attain the project's goals.

The expected outcome was that two distinct but cooperating teams would produce a code base with many similarities, while two competing teams would produce clearly distinct code bases. The proposed method for answering this question is to use the same approach as in RQ1, but applying it to the two data sets obtained from code measurements and team measurements, thus estimating a "code-base tree" and a "team tree" for each fork.

Null hypothesis
The null hypothesis is that forks with different outcomes are indistinguishable based on code and organizational measurements.

Alternative hypothesis
The alternative hypothesis is that the outcome of a fork can be predicted by measuring the pairwise distances between releases, based on separate code and organizational data sets.

These hypotheses can be translated into statistical hypotheses by calculating the variance of the cophenetic distances between trees of forks obtained from code measurements and trees constructed from team measurements.

Statistical null hypothesis
The statistical null hypothesis is that the mean distance between forks with different outcomes are not significantly heterogeneous with regard to the outcome of the fork.

Alternative statistical hypothesis
The alternative statistical hypothesis is that the mean distance between forks with different outcomes are significantly heterogeneous with regard to the outcome of the fork.

3.3.3 Research procedures

The research questions are examined by applying the activities detailed in figure 3.1.

Data acquisition and re-encoding

Data was acquired from selected open source repositories. The choice of software projects to study was guided by the comprehensive review of forks provided by Robles and González-Barahona (2012). Forks are per definition only possible in an open source context, therefore any conclusions apply in this context only. Additionally, the choice of projects was limited by the availability of a complete release history of the project, at least as far back as the moment in time were the fork happened.

As forking is a process at the crossroads between technical evolution and organizational evolution (as per the definition in paragraph 1.3), the data acquisition techniques were chosen to cover technical and organizational aspects of projects.

Data has to be re-encoded in order to be processable. This process is explained in detail in paragraph 4.1.2. Guidelines for re-encoding characters can be found in Sokal (1983).

Distance matrices

As explained in paragraph 3.1.2, the chosen methods for answering the research questions are all part of the "distance matrix" family of phylogenetic methods. As shown in figure 3.1, a matrix of the pairwise dissimilarity between the releases of the chosen fork has to be computed. Several techniques are available to compute a distance matrix: computing Euclidean, or "Manhattan" distances are commonly used (Felsenstein, 1982). However, these techniques are limited to processing numerical and Boolean data respectively. As the acquired data is of various data types (Boolean, categorical, numerical, see table 3.1) the chosen method is required to process different data types. Only the "Gower" technique (D'Orazio, 2016) satisfies this requirement. However, the Gower technique is limited in that matrix rows with missing values do not contribute to the result, therefore, part of the data acquired was discarded before processing.

Table 3.2 shows an example distance matrix, constructed by subsampling the data collected for the Mysql / MariaDB fork (see 4.1 for details). Ten branches and thousand measurement characters were selected at random (the original data set has 107 branches and 56 648 measurement characters).

Phylogenetic trees and cophenetic distances

The proposed methods for answering the research questions involve constructing phylogenetic trees by applying the methods described in 3.1.3.

Figure 3.2 shows an example tree constructed using the subsample of the data presented in table 3.2. The Neighbour-Joining algorithm (Saitou and Nei, 1987) was used to estimate the tree.

The next step is to calculate the cophenetic distances between the tips of the trees. Figure 3.2 illustrates this: the green line represents the shortest path between two tips in the example; the red line represents the longest path between two tips.

Table 3.3 shows an example cophenetic matrix. Please note that the cophenetic matrix is calculated from the tree, so the method used to estimate the tree has an effect on the cophenetic matrix. Conversely, the correlation between the distance matrix and the cophenetic matrix can be used as a measure of how accurately the tree represents the distance matrix.

The distance matrix can be shaded to show the degree of dissimilarity between releases, and the matrix can then be rearranged into clusters of the same shade. This representation was introduced by Sokal (1962) and used to discover species and genus in the data. Table 3.4 shows that in the example data, MySQL releases are close to each other, while MySQL 5.7.12 and all releases of MariaDB are the most distant from each other (light-shaded cells in the bottom row).

Analysis of variance

As detailed in 3.2.1, answering RQ1 requires computing a matrix of the cophenetic distances between releases and estimating if these distances correlate to forks and branches. Figure 3.3 shows a box plot of the cophenetic distances in the example data. The plot shows that distances between tips inside either project ("Branches") are statistically smaller than the distances between tips on different forks ("Forks").

According to McDonald (2014), choosing a method for statistical analysis is primarily guided by the kind and number of variables required to test the hypothesis. The cophenetic distance is a measurement variable, and whether the pairwise

distance between releases corresponds to a "fork" or "branch" is a nominal variable. McDonald (2014) lists two methods for testing a hypothesis based on one measurement and one nominal variable: "student's t-test" and "analysis of variance" (ANOVA).

The method proposed for answering RQ2 (paragraph 3.2.2) requires examining projects with different outcomes and computing two cophenetic distance matrices for each project: one for the tree obtained from team measurements, and a second one for the tree obtained from code measurements. Having computed these matrices, the next step is to estimate if these distances correlate with different project outcomes. The cophenetic distance is a measurement variable, the data set used ("team measurements" or "code measurements") is a nominal variable and the outcome of the project ("cooperation", "competition" or "discontinuation of a branch") is a second nominal variable. McDonald (2014) recommends applying a two-way ANOVA for testing a hypothesis based on one measurement and two nominal variables.

Consequently, RQ1 and RQ2 can be answered by applying analysis of variance (ANOVA), albeit with different number of nominal variables. ANOVA assumes that the data is normally distributed (fits the Gaussian "bell curve") and that that the standard deviation is the same for the data sets examined, otherwise false positives may result (McDonald, 2014. p. 147). As the raw data obtained from the repositories did not meet these assumptions, the data had to be transformed to fit the assumptions prior to the analysis.

3.3.4 Ethical considerations

All data analysed for this research is freely available under an open source license, therefore no intellectual property rights were disregarded during the acquisition of the data.

Some of the forks studied were a consequence of disagreements between development teams. Care was applied during the analysis not to take sides for any particular team; however, to thwart any bias ensuing from the reputation of well-known actors in the industry, all names of developers were anonymized prior to being stored locally.

Although I do participate in open source software development, I am not involved in any of the projects examined for this research.

3.3.5 Summary of Chapter 3

Character measurement techniques applicable to software code bases and to software producing organizations were selected from literature. Suitable techniques to study the evolution of software were chosen from the corpus of biological techniques for estimating phylogenetic trees. The two research questions were specified in terms of hypotheses, null hypotheses and their corresponding statistical hypotheses. The steps required by the research were worked out and each step was illustrated with an example drawn from a subset of the data. Finally, analysis techniques suitable to answer the research questions were selected, based on the type and number of independent variables required by the analysis.