

Evolution of Collaboration in Open Source Software Ecosystems

Jos van der Maas
j.c.vandermaas@students.uu.nl



Utrecht University

Department of Information and Computing Sciences
Master in Business Informatics

January 2016

MSc thesis under supervision of:

First supervisor: Dr. S. Jansen (Utrecht University)

Second supervisor: Dr. F.J. Bex (Utrecht University)

Abstract

Although in the past decennium much research has been conducted in the field of software ecosystems, still little is known about how such ecosystems evolve. In existing literature, software ecosystems are often analyzed from a single point of time, without regard for their dynamic history.

We study an aspect of the evolution of open source software ecosystems: collaboration. In this thesis, we present a method to identify and analyze collaboration networks in software ecosystems, based on historical data from the online source code hosting service GitHub. We apply this method to analyze the evolution of collaboration within the well-established open source ecosystem around the Ruby programming language.

Our method can be used to identify and visualize collaboration networks around a given actor or software component over the course of time. This is done using statistical analysis techniques, without requiring subjective input. Our study analyzes over 200 collaboration networks to identify typical life cycle shapes and gives insight in the evolution an open source software ecosystem. Awareness of such collaboration networks and how they evolve is beneficial for actors in software ecosystems and for external stakeholders.

Keywords: software ecosystem(s), collaboration, network(s), life cycle(s), GitHub, Ruby.

Acknowledgements

This Master's thesis is the result of a research project that took several months. Looking back at this period, I am thankful to several people.

First of all, I would like to thank my first supervisor Slinger Jansen for his time, his quick and constructive feedback, and his insightful recommendations. Slinger, thanks for your supervision and the directions you gave by email, Skype, and face-to-face. I could not have wished a better supervisor.

In the second place, I would like to thank my other supervisor Floris Bex for his feedback and for reviewing this thesis document. I would also like to thank the other staff members of the Center for Organization & Information for the MBI study program, which I really appreciated.

I would like to thank my colleagues for allowing me to work part-time while doing my thesis research, my parents for their interest in my study progress, my wife for her support and our five months old son for sleeping well at night and allowing me to rest enough to finish this thesis project.

Contents

1	Introduction	1
1.1	Problem Statement	3
1.2	Research Questions	6
1.3	Relevance	7
1.3.1	Scientific Relevance	7
1.3.2	Practical Relevance	7
1.4	Document Structure	8
2	Research Approach	9
2.1	Literature Review	9
2.2	Data Collection and Processing	11
2.3	Data Analysis	11
2.4	Plan Validity	11
2.4.1	Construct Validity	12
2.4.2	Internal Validity	12
2.4.3	External Validity	12
2.4.4	Reliability	13
3	Literature Review	14
3.1	Definition of Software Ecosystems	14
3.2	Relationships in Software Ecosystems	15
3.3	Network Perspective	17
3.3.1	Node Level Analysis	17
3.3.2	Network Level Analysis	19
3.3.3	Visualization Frameworks and Tools	21
3.4	Evolution of Software Ecosystems	21
3.4.1	Life Cycles	21

3.4.1.1	What to measure	22
3.5	Mining Software Repositories	23
4	Mining GitHub Data for Collaboration Networks	26
4.1	Collaboration Network Identification and Measurement	26
4.2	Data Collection	29
4.3	Data Processing	30
4.4	Data Analysis	30
4.5	Results	31
5	Categories of Open Source Collaboration Networks	32
5.1	Procedure	32
5.2	Hypotheses	36
5.3	Results for User-Centered Collaboration Networks	36
5.4	Results for Repository-Centered Collaboration Networks	38
6	Characteristics of Collaboration Network Categories	40
6.1	Category A: Short Revival Before Abandonment	42
6.2	Category B: Early Maximum	42
6.3	Category C: Extended Growth	43
7	Collaboration in the Ruby Ecosystem	44
7.1	Observations	45
7.1.1	Late 2013	50
7.1.2	Early 2014	50
7.2	Summary of Observations	52
8	Discussion	53
8.1	Findings and Implications	53
8.2	Validity	57
8.3	Future Research	57
9	Conclusion	59
	Bibliography	60
	Appendix A: Analysis Data	65

Chapter 1

Introduction

“If you think the industrial revolution was transformational, the App Store is way bigger”.

This statement appeared in a video¹ aired at Apple’s World Wide Developers Conference in June 2015. Although at first sounding like exaggerated marketing talk, the remark originates from the director of the McKinsey Global Institute, a research group concentrating on major economic trends around the world.

Many other scientific sources acknowledge the impact of the introduction of app stores, which are marketplaces for applications that can be instantly downloaded and installed on a customer’s device. Jansen & Bloemendal (2013) call it “one of the most powerful changes that the software business currently is experiencing” and Miluzzo, Lane, Lu, & Campbell (2010) speak about “a new era”.

In a broader sense, we observe that, especially since the introduction of the Internet, software is often built around a certain technological platform or market it interacts with (Bosch, 2009; Molder, Van Lier, & Jansen, 2011). Examples such technological platforms or markets are: app stores for mobile applications (such those facilitated by Apple, Google and Microsoft), the Ruby programming language with its network of ‘gems’ that often depend on other Ruby gems², and online CMSes like Wordpress³, Joomla⁴, and Drupal⁵, each with its own community of users and numerous available plugins.

In general, we observe that “networks of interrelated organizations form themselves around products, platforms, technologies or software organizations” (van Angeren, 2013). Such networks are referred to as *software ecosystems* (Messerschmitt & Szyperski, 2003). Jansen, Finkelstein, & Brinkkemper (2009) define a software ecosystem as

“a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts”.

¹<https://www.youtube.com/watch?v=fSiDIaab2nY&t=72>

²See <https://rubygems.org/>

³<https://wordpress.org/>

⁴<http://www.joomla.org/>

⁵<https://www.drupal.org/>

Other definitions differ from the above one by putting more emphasis on relationships between software components. The choice of definition determines how software ecosystems are analyzed and visualized.

Several authors have suggested methods to visualize software ecosystems, mostly by depicting ecosystems as networks. However, there is no widely accepted visualization method yet (Boucharas, Jansen, & Brinkkemper, 2009; Goeminne & Mens, 2010; Campbell & Ahmed, 2010; Pérez, Deshayes, Goeminne, & Mens, 2012).

Open source software ecosystems are especially suitable for research. Version control systems such as Git⁶ and Subversion⁷ keep exact records of changes in program code, in addition to information about authors and software dependencies. This data is often publicly available, making it possible to analyze open source ecosystems based on objective data. Collaboration is a key aspect of open source software, which can be measured by mining software repositories, from which collaboration networks can be identified. These networks can then be analyzed using scientific techniques from graph theory.

In recent years, researchers have made the connection between software ecosystems and (product) life cycles (dos Santos & Werner, 2011; Kim, Lee, & Altmann, 2014). Such research aims to capture life cycles of software ecosystems, describing various typical stages, such as emergence, maturity and downfall.

To study life cycles of collaboration networks in software ecosystems, longitudinal research is required. For many open source ecosystems however, a detailed history of changes is already publicly available, including timestamps, author information, and information about the content of the changes. This opens possibilities to study life cycles of collaboration networks based on historical data, without having to follow the actors in the networks for a longer period of time.

⁶<https://git-scm.com/>

⁷<https://subversion.apache.org/>

1.1 Problem Statement

Still much is unclear about the evolution of software ecosystems. This problem can be broken down into three or four sub-problems, as described below.

Static view

In existing scientific literature, the state of a software ecosystem is often observed from one specific point in time, instead of being regarded as a continuous evolutionary process. For example, Kabbedijk & Jansen (2011) give a detailed network visualization of the Ruby ecosystem. However, the visualization shows the status of the ecosystem for one point in time, namely the time when the research was carried out. The authors acknowledge that “due to the lack of longitudinal data it is impossible to speculate about the dynamics of the Ruby SECO [software ecosystem]”.

A similar situation is found in the study of Blincoe, Harrison, & Damian (2015). This publication gives a visualization of multiple software ecosystems for which the code is hosted on GitHub, but again only for a single point in time.

In this respect, van Angeren (2013) remarks that

“a longitudinal case study of proprietary platform ecosystems could benefit a wealth to the understanding of the fast-paced dynamics characteristic for the software industry”.

Similarly, Kim, Altmann, & Lee (2013) remark that

“prior research concentrates only on the static properties of network structure and the position of nodes in the network, but misses the dynamics in the evolution context”.

Insight in this “dynamics in the evolution context” is important for our understanding of software ecosystems.

Validity of life cycle models

While there exists some scientific theory on software ecosystem evolution and especially on ecosystem life cycles, this theory needs to be validated. For example, dos Santos, Esteves, Freitas, & de Souza (2014) suggest that a software ecosystem life cycle goes through the consecutive phases of Initiation, Propagation, Amplification, and Termination, following a curve as shown in Figure 1.1.1. Kim, Lee, & Altmann (2014) mention three phases: Ascent/Emergence, Maturity/Prosperity, and Decline.

The theory of ecosystem size following a path along consecutive stages originates from product life cycle theory, such as the work of Polli & Cook (1969) and Tellis & Crawford (1981). However, it is unclear whether this theory can be readily applied to the dynamic nature of software ecosystems. Such an assumption must be tested in practice.

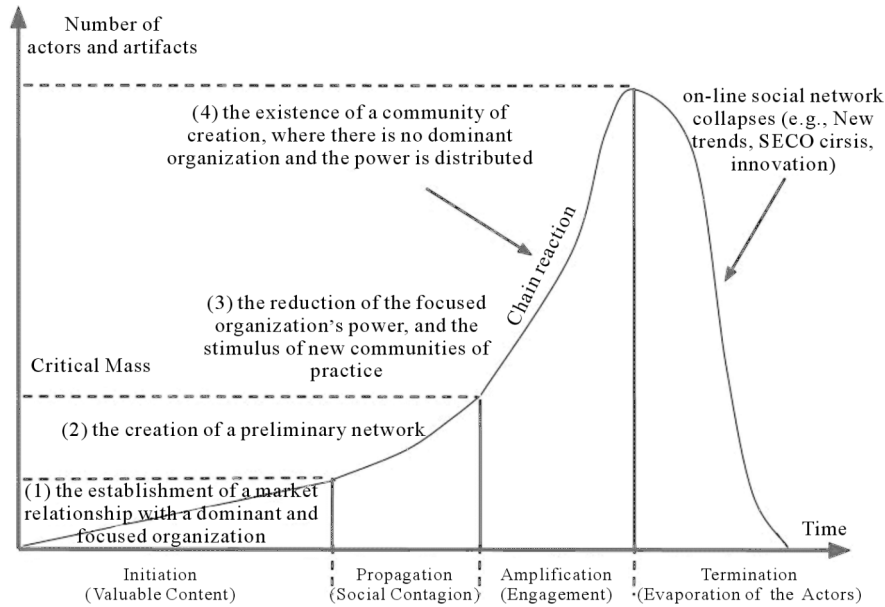


Figure 1.1.1: Software ecosystem maturity model proposed by dos Santos et al. (2014).

Small samples

Another problem is that existing analysis of software ecosystems is often based on relatively small samples, i.e. a small number of networks that is analyzed per publication. It would be interesting to analyze and compare a large number of networks to each other.

Many publications are limited to only one well-established software ecosystem, such as Eclipse (Dhungana, Groher, Schludermann, & Biffi, 2010), Ruby (Kabbedijk & Jansen, 2011), or Apache (Santana & Werner, 2013). Few studies try to compare data from multiple ecosystems to each other.

Domain-specific measurement of ecosystem relationships

It is debatable how relationships in software ecosystems should be identified. Going back to the definition of Jansen et al. (2009), we see that a software ecosystem is “a set of actors (...), together with the relationships among them”. Other definitions of software ecosystems are similar in the sense that they tell us that there are some kind of ‘relationships’ between actors or software components. However, in what way these relationships can be tracked or measured is subject to wide interpretation.

In existing literature, data for measurement of relationships in software ecosystems is often collected from a domain-specific source, such as the Gnome website⁸ for the Gnome ecosystem (Mens & Goeminne, 2011) and the RubyGems website⁹ for the Ruby ecosystem (Kabbedijk & Jansen, 2011). However, the way in which such data is collected differs and is often difficult to generalize for other ecosystems.

⁸<https://www.gnome.org/>

⁹<https://rubygems.org/>

Summary

In short, these issues can be captured in the following problem statement:

It is unclear how collaboration in open source software ecosystems evolves. Existing research fails to longitudinally analyze a substantial amount of collaboration data in a generalizable way.

We will narrow our research scope to *open source* software ecosystems, as reflected in this problem statement.

Approach outline

In order to address the stated problem, we propose to use historical data from GitHub¹⁰, which is the largest online open source code repository (Gousios, Vasilescu, Serebrenik, & Zaidman, 2014), as data source. This data contains precise information about changes in program code of thousands of open source projects. The data is publicly available and can be queried to reconstruct time-specific collaboration networks. These networks are then analyzed to study the evolution of collaboration in open source software ecosystems.

¹⁰<https://github.com/>

1.2 Research Questions

The main research question addressed in this thesis is:

How does collaboration in open source software ecosystems evolve?

In order to answer this research question, we formulate the following sub research questions:

1. **What are relationships in software ecosystems?** The way in which relationships in software ecosystems are measured is of crucial importance for the further analysis of these ecosystems. In the past, researchers have chosen to base relationships on, amongst others, co-authorship (Kabbedijk & Jansen, 2011), API implementation (Kim, Lee, & Altmann, 2014), and the use of cross-references (Blincoe, Harrison, & Damian, 2015). Furthermore, the information sources for the determination of such relationships vary from mailing lists (Bird, Gourley, Devanbu, Gertz, & Swaminathan, 2006) to domain-specific websites (Hoving, Slot, & Jansen, 2013), surveys (Syed & Jansen, 2013), and company websites (van Angeren, Alves, & Jansen, 2014). We study similarities and differences between these determination methods in our literature review and propose a definition of software ecosystem relationships.
2. **How can collaboration in software ecosystems be measured and visualized, based on data from a public data source?** We query historical data from a public data source (GitHub) to identify collaboration networks and measure their state. The state of a network can be measured using several properties. For example, van Angeren et al. (2014) use the properties Size, Density, Degree centrality, Centralization, Modularity, and Clustering. We calculate similar properties for collaboration networks on a day-to-day basis to analyze their life cycles.
3. **Can various categories of collaboration networks be distinguished, based on their life cycles?** An important question is to which extent (product) life cycle stages such as Initiation, Propagation, Amplification, and Termination are applicable to collaboration in open source ecosystems. Our hypothesis is that the life cycles of certain collaboration network fit these stages, while others have a completely different cycle. We aim to discover a concise number of categories of collaboration networks that resemble each other in terms of their life cycles. Using a sample set of randomly selected networks, we study such categories.
4. **How are the collaboration networks in these categories characterized?** Once we have established a number of life cycle categories, we study which characteristics or properties are typical for these categories. For example, are there certain deviations in the network structure of collaboration network that are a good indicator for the approximate life cycle course?
5. **How does collaboration in an open source ecosystem evolve in practice?** This question is addressed by a case study on the ecosystem around the Ruby programming language.

To provide an answer to the (sub) research questions, data from the GitHub Archive¹¹ (a project that maintains a history of GitHub data) will be gathered in a database and a tool will be developed that can

¹¹<https://www.githubarchive.org/>

identify collaboration networks, starting from a single project or author. The tool will then be used to compare and analyze a large number of networks using statistical analysis techniques.

1.3 Relevance

1.3.1 Scientific Relevance

Due to its recentness, the research domain of software ecosystems leaves opportunities for research unaddressed (Barbosa & Alves, 2011; Manikas & Hansen, 2013). Although relatively much research has been conducted on open source ecosystems and modeling (Barbosa & Alves, 2011), the research field of software ecosystem evolution is still largely unexplored.

A large proportion of the literature on software ecosystems consists of case studies on single predefined ecosystems. Limited attempts have been made to conduct a thorough quantitative analysis, comparing a significant number of networks to each other.

Longitudinal studies on software ecosystems are rare. Our research will give insight in ecosystem evolution and life cycle categories. Theory about life cycle maturity stages will either be substantiated or disproved. The results of our research could be useful for prediction of evolution of ecosystems in their early stages.

The tool that is developed as part of the research is useful for further in-depth research, e.g. about actor roles and their impact on the evolution of ecosystems.

1.3.2 Practical Relevance

Apart from its scientific relevance, our research has several benefits for practitioners.

Our study will facilitate the identification and exploration of open source ecosystems. As Blincoe et al. (2015) state in their research agenda: “A tool could be developed that automatically identifies technical dependencies across projects and provides a visualization of the ecosystem. Such a tool could increase awareness of coordination needs that extend outside project boundaries and help developers gain a better view of the ecosystem surrounding their project.” Awareness of the ecosystem around a software project can greatly benefit its developers.

For facilitators of ecosystems (such as the Eclipse Foundation¹²), insight in the development of ecosystems would be of great value to help stimulate the growth and increase the maturity and robustness of their ecosystem.

Furthermore, this research provides insight into the characteristics of different categories of collaboration networks. Once known which characteristics should be pursued and which should be avoided, strategical decisions can be made in the development of software projects.

¹²<https://eclipse.org/org/foundation/>

1.4 Document Structure

After its introduction in Chapter 1, this thesis continues with an explanation of the research approach in Chapter 2. The next chapter contains the theoretical background for our research, answering our first research question.

Chapter 4 describes the development of the analysis tool and aims to answer the second research question. After this, Chapter 5 describes categories of collaboration networks and contains the results of the analysis of research question 3. Chapter 6 addresses the fourth research question, describing characteristics of software ecosystem categories.

The analysis part from Chapters 3 to 6 is followed by a case study of the Ruby ecosystem in Chapter 7, a summary and discussion of the research findings in Chapter 8, also containing suggestions for future research. The thesis ends with a conclusion in Chapter 9.

Chapter 2

Research Approach

Our research approach can be summarized as follows:

- Conduct a literature study on software ecosystems in general and software ecosystem relationships, characteristics, visualization, and analysis.
- Use GitHub and the GitHub Archive¹ as data source.
- Define a method for measuring relationships and use this to identify collaboration networks for an arbitrary repository or developer as origin of the network.
- Develop a tool that can query the data to visualize the state of a collaboration network on a certain point in time.
- Let the tool follow the state of collaboration networks over time to obtain graphs depicting their life cycles.
- Define sample sets of open source collaboration networks on GitHub.
- Statistically analyze the life cycles of the networks in the sample set(s) to identify categories of collaboration networks, based on their life cycles.
- Analyze the network properties of the categories to identify common network properties.
- Conduct a case study on the Ruby ecosystem to analyze how collaboration in ecosystems evolves in practice.

2.1 Literature Review

In this section, we describe the approach to conduct our literature review.

¹<https://www.githubarchive.org/>

Method

We will use a scoped literature review, as described by Arksey & O'Malley (2005). In case of a scoped literature review (as opposed to a systematic literature review), a number of concrete consecutive steps can be followed to obtain a theoretical background for a research project. A scoped literature review aims to obtain information about a relatively broad topic, without seeking to address a specific study design or answer very specific research questions. As a result, there is less need to assess the quality of included studies. Scoped literature reviews can be also used to identify research gaps in existing literature (Arksey & O'Malley, 2005).

Systematic literature reviews for the research domain of software ecosystems have been conducted by Barbosa & Alves (2011) and Manikas & Hansen (2013). These studies revealed that the attention for software ecosystems is increasing, resulting in a large number of available publications on this topic.

Sources

To retrieve relevant sources for the literature review, our collection process consists of a number manual search queries for scientific contributions. Searches were performed on the scientific databases of ACM², Emerald³, IEEE⁴, Mendeley⁵, ScienceDirect⁶, Springer⁷, and Wiley⁸. The following keywords were combined into search strings:

("software ecosystem" AND ("relationship" OR "network" OR "analysis" OR "evolution" OR "life cycle" OR "lifecycle" OR "visualization" OR "visualisation" OR "trend" OR "classification")) OR "mining software repositories"

Where applicable, both the singular and plural forms of keywords were used.

The following inclusion criteria were used to select sources for the literature review:

- All sources should be peer-reviewed scientific contributions.
- All sources should be written in English.

To retrieve additional scientific sources, bibliographies of initially selected sources were reviewed. Relevant sources were considered based on their title, keywords, and abstract.

Relationships

In our literature review, we analyze how other scientific sources determine relationships. Based on the results from this literature review, we formulate our own definition of software ecosystem relationships.

²<http://dl.acm.org/>

³<http://www.emeraldinsight.com/>

⁴<https://www.ieee.org/>

⁵<https://www.mendeley.com/research-papers/>

⁶<http://www.sciencedirect.com/>

⁷<http://link.springer.com/>

⁸<http://onlinelibrary.wiley.com>

2.2 Data Collection and Processing

Based on our definition of software ecosystem relationships, we define a method for identifying open source software ecosystems on GitHub. We limit our research to open source software ecosystems. An advantage of analyzing open source software is that the data is transparent, with much information being publicly available. Another advantage is that much of this information is objective, as opposed to e.g. survey data.

A key characteristic of open source software is that it is generally developed by multiple developers at the same time. Therefore, we focus at collaboration relationships. We will analyze both relationships between software developers (actors) and software components.

Our goal is to analyze how software ecosystems develop over a period of time and to compare various stages in the life cycles of collaboration networks. For this, we need historical data, which we obtain by mining data of software repositories on GitHub.

Collaboration information on GitHub is stored in the form of commits, which can be pushed or requested to be pulled. Therefore, we analyze data from push events and (merged, i.e. successful) pull request events. To import and analyze the data, we build a web-based tool, which should at least have the following functionality:

- Importing data required for our analysis
- Establishing collaboration networks, based on co-authorship and starting from an arbitrary user or repository
- Visualizing these networks, over a period of time
- Visualizing network properties of these networks, based on literature from the literature review

2.3 Data Analysis

Based on our findings in the literature review, we obtain a sample set of collaboration networks and analyze how each of these networks evolves. We categorize the collaboration networks based on their life cycle shapes. We statistically test which life cycle shapes are most common. We also calculate the average life cycle for an open source ecosystem, based on our sample set. We do this for both repository-centered life cycles and user-centered life cycles.

We compare the most common life cycle categories to each other in terms of several network properties. We statistically test the differences between these categories.

After this, we analyze collaboration in a predefined open source ecosystem. We apply our analysis method to the Ruby ecosystem to see how an ecosystem evolves in practice over the course of several years.

2.4 Plan Validity

The validation of our research approach can be divided in four types of validity types.

Construct validity indicates that correct operational measures are used for the concepts that are studied.

Internal validity is used for making sure that the casual relationships (where applicable) are properly linked.

Research is said to be *externally valid* if the findings are generalizable beyond the sample set. *Reliability* indicates that if someone else wants to continue our work, this should be possible. In other words, should someone else follow the same procedures as we did and conduct the same case studies, then the findings and conclusions should be the same.

2.4.1 Construct Validity

We propose to observe evolution of open source software ecosystems by measuring co-authorship based on push and (merged) pull request events. Instead of co-authorship, we could e.g. use email addresses or organization membership. Similarly, instead of pushes and pull requests, we could measure posted comments, opened issues, etc.

An advantage of looking at co-authorship is that this represent a natural connection in the real world, namely that of co-creation. Moreover, this relationships can be objectively measured. An advantage of using push and pull request evens is that the results do not depend on how much information a user specifies. E.g. email addresses can be omitted to prevent spam and organization membership might not (intentionally or unintentionally) be specified by a developer.

A limitation of our choice of measurement is that we do not measure the volume or impact of these events. E.g. the number of altered lines of code can vary widely per push and pull request event. However, on average the number of such events per user can be assumed to be a good indicator for the user's activity. When many users make many pushes or pull requests, the ecosystem can be called alive and successful. When no more such events occur in a software ecosystem for several months (e.g. three months), the ecosystem can be assumed to have ended.

We will consider push and pull data per day. Instead of days, we could consider time intervals of hours, weeks, months, etc. However, using longer intervals (e.g. weeks) would result in relatively few data points, since we only have the years 2011-2015 as scope. Using shorter intervals (e.g. hours) would require significantly more computation power and could cause strong deviations caused by working hours, time zone differences, etc. As such, the choice of days as time intervals is appropriate and can be assume not to bias the results.

2.4.2 Internal Validity

Since the data is obtained from a public data source in retrospect, users do not know they are being studied for this research. Thus we can assume that they are not influenced by the measurement. We will use random selections to compose sample sets.

Life cycles of collaboration networks can be influenced by events outside our measurements, such as the temporary absence or death of a developer. However, such events can be regarded as part of the dynamics of software ecosystems.

2.4.3 External Validity

When it comes to external validity, our research has various limitations due to the choice of GitHub as data source:

- GitHub was launched in April 2008 and is therefore relatively new. However, it is currently the largest code host in the world, facilitating millions of developers collaborating across millions of repositories, according to Gousios, Vasilescu, Serebrenik, & Zaidman (2014).
- We have access to data of open source projects only. It is questionable whether or not this can be generalized to non-open source ecosystems. However, we limit our scope to open source software ecosystems.
- GitHub is not the only platform that facilitates open source software ecosystems. Not everything is hosted on GitHub. As Kalliamvakou, Gousios, Blincoe, Singer, German, & Damian (2014) write, “Many active projects do not conduct all their software development in GitHub”.
- GitHub only works with one version control system (Git, which was introduced in 2005).
- We obtain our data from the GitHub Archive project. This project has recorded data from February 12th, 2011 onwards.
- Since we look at complete life cycles, we are limited to life cycles that fall within a the time period from early 2011 till mid 2015, a period of four years.
- It is possible that projects are hosted on GitHub that have existed prior to their submission to GitHub (e.g. projects that earlier used Subversion for version control). This will result in misleading activity numbers at the ‘start’ of such a software projects. However, since we only have data from 2011 and onwards and since GitHub already existed in 2008, these effects will likely be relatively low. Also, one large addition of code is only counted as a single event.
- Some features of GitHub, like organization membership, have not existed from the start of GitHub. But we will most likely not use these relatively new features for our research.
- In the research period, GitHub and the GitHub Archive Project had some downtime (i.e. were unaccessible). However, from our database results it is clear that this downtime was relatively short and its effect insignificant for the results.

Despite these limitations, our approach makes it possible to study and compare a relatively large number of collaboration networks. Other studies have comparable limitations. Therefore we are confident that our research will shed new light on the evolution of open source software ecosystems.

2.4.4 Reliability

Part of our research will be the development of the tool to visualize and measure collaboration networks. This tool will be web-based, publicly accessible, and submitted as an open source project to GitHub. This tool can be used to verify our research and for further research.

Chapter 3

Literature Review

The concept of *software ecosystem* was introduced by Messerschmitt & Szyperski (2003) and has since been the subject of extensive research. During the past decade, the number of scientific publications on Software Ecosystems (SECOs) has significantly increased (Barbosa & Alves, 2011; Manikas & Hansen, 2013).

Our literature review will focus on the following subjects:

- Definitions of software ecosystems, in order to know precisely what we study.
- Relationships in software ecosystems, to be able to analyze ecosystems as networks.
- Network theory, to measure the state of the networks.
- Theory about evolution of software ecosystems, to help study our main research question.
- Mining software repositories, to gather data for analyzing open source software ecosystems.

3.1 Definition of Software Ecosystems

Messerschmitt & Szyperski (2003) define a software ecosystem as

“a collection of software products that have some given degree of symbiotic relationships”.

While this to our best knowledge is the earliest formal definition of software ecosystems, the most cited definition is (according to Manikas & Hansen (2013)) that of Jansen, Finkelstein, & Brinkkemper (2009), who define a software ecosystem as

“a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts”.

Other definitions that are commonly referred to are that of Bosch (2009):

“a set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions”

and the definition given by Lungu et al. (2010):

“a collection of software projects which are developed and evolve together in the same environment”.

While these definitions vary, Manikas & Hansen (2013) remark that the definitions of software ecosystems to some extent contain the following three common elements:

1. A common software platform or environment;
2. A shared business or community; and
3. Connecting relationships.

The choice of definition has consequences for the perception of software ecosystems. In this thesis, we use the definition of Jansen et al. (2009), with the side note that relationships in software ecosystems can also be regarded at a software level, as further explained in the next section.

3.2 Relationships in Software Ecosystems

When analyzing software ecosystems, it is important what exactly one calls a relationship. In the definitions of software ecosystems we already observe differences. Some definitions speak of relationships *between actors* (e.g. the definition of Jansen et al.) and others of relationships *between software components* (e.g. the definition of Messerschmitt & Szyperski). These different views have consequences for the analysis and visualization of software ecosystems.

A number of examples from literature of what researchers choose as relationships is given below.

- Crowston & Howison (2005) base relationships between developers of open source projects on email addresses found in bug reports on public mailing lists.
- Bird, Gourley, Devanbu, Gertz, & Swaminathan (2006) base relationships between developers of open source software on public emails sent among them.
- Kabbedijk & Jansen (2011) base relationships in the Ruby ecosystem on co-authorship data from rubygems.org.
- Syed & Jansen (2013) base relationships between developers in the Ruby ecosystem on collaboration information from rubygems.org and a survey sent to a large number of developers.
- Hoving, Slot, & Jansen (2013) base relationships in the Python ecosystem on collaboration data from python.org.

- van Angeren, Alves, & Jansen (2014) base relationships between companies in the Google Apps software ecosystem on information found on company websites and on CrunchBase.com.
- Kim, Lee, & Altmann (2014) base relationships between providers of APIs and mashups that use those APIs on data from programmableweb.com.
- Gregorian (2014) bases relationships between GitHub developers on a complex formula that combines several forms of communication between the developers.
- Blincoe, Harrison, & Damian (2015) base relationships between GitHub repositories on cross-references between those repositories.

From these sources, we draw a number of conclusions:

1. Many different choices seem to be suitable to use as relationships. The concept of relationship is not strictly defined.
2. The essence of all relationships is the exchange of information. There must be some interaction or information exchange through the relationship.
3. Information exchange can exist between both actors (e.g. via emails) and software components (e.g. via an API).
4. In many cases, there is some common system or platform that facilitates the exchange of information (e.g. a mailing list or a code repository). Data from such a platform can be used to measure relationships. This platform is often facilitated by what is called in literature a keystone (player). The information exchange can also be facilitated by a common technology, such as a software project or an API.
5. Relationships (in general) have the properties continuation and strength (Holmlund, 1997). The relationships can be measured in terms of these properties.
6. The relationships are subject to continuous change because of their dynamic nature.
7. Relationships have potential because they provide access. They must be viewed in their context, in this case the ecosystem in which they are embedded.

From these conclusions, we formulate our own definition of software ecosystem relationships:

Definition. A *relationship* in a software ecosystem is a measurable dynamic form of information exchange between either two actors or two software components in the ecosystem. Such information exchange is often facilitated by a common technology or platform.

We can distinguish between explicitly measurable connections (e.g. commits to a software repository) and relationships that are open to human interpretation (e.g. surveys or information found on company websites).

We observe that researchers often use an ecosystem-specific data source to measure relationships. This makes it less suitable to compare data from different software ecosystems to each other.

3.3 Network Perspective

Observing software ecosystems as networks consisting of nodes and links opens possibilities for research and has two clear benefits. Firstly, networks provide a useful metaphor to communicate and interpret ecosystems and secondly, networks provide a basis for analytics through mathematical graph theory.

In literature, software ecosystems are often modeled as networks in the form of undirected graphs, implicating that the relations are symmetric (the relationship from A to B is the same as that from B to A). In our analysis, we use weighted undirected graphs, i.e. each relationship is given a numerical weight.

To analyze collaboration networks in software ecosystems, we use a number of well-established measures from graph theory. This is done on node level (focusing on the role of single nodes) as well as on network level (focusing on the state of a network as a whole).

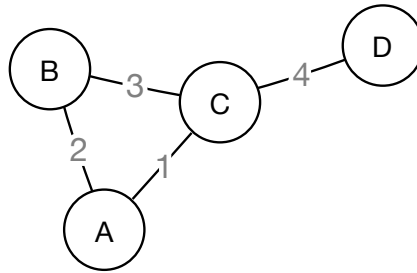


Figure 3.3.1: Example of an undirected weighted graph that could represent a Software Ecosystem consisting of nodes A, B, C, and D and the relationships among them.

3.3.1 Node Level Analysis

Degree

The *degree* of a node is defined as the number of edges connected to the node.

Formally defined, consider a graph with n nodes and *adjacency function* a , meaning that for two given nodes p_i and p_k in the network,

$$a(p_i, p_k) = \begin{cases} 1 & \text{if and only if } p_i \text{ and } p_k \text{ are connected by an edge} \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$\text{degree}(p_i) = \sum_{k=1}^n a(p_i, p_k).$$

For example, node A in Figure 3.3.1 has degree 2, since it is connected to node B and node C only.

Weighted Degree

The *weighted degree* of a node is the sum of the edge weights of its connecting edges. For example, node A in Figure 3.3.1 has a weighted degree of $2 + 1 = 3$. Node D has a lower degree than node A , but its weighted degree is higher.

Degree Centrality

The *centrality* of a node is a measure of how central a node in a network is compared to the other nodes. For example, in Figure 3.3.1 intuitively node C seems to be the most central node in the network. Freeman (1978) defines the *degree centrality* of a node p_k in a graph with n nodes as follows.

$$\text{relative degree centrality}(p_k) = \frac{\text{degree}(p_k)}{n - 1}.$$

Note that this is essentially the same as the degree of the node divided by the maximum degree it could possibly have (which is $n - 1$, namely in case it is connected to all other nodes in the network). For example, the relative degree centrality of node C in Figure 3.3.1 is $\frac{4}{5-1} = 1$, since this node is connected to all the other nodes in the network. Since the other nodes have lower degree centralities, node C can indeed be called the most central node in the network.

Clustering Coefficient

Watts & Strogatz (1998) define the (local) *clustering coefficient* of a node as the number of edges between the nodes in its neighborhood, divided by the number of edges that could possibly exist between them. Here, the neighborhood of a node refers to nodes that are its direct neighbors, i.e. adjacent nodes.

For an undirected network, if a given node p_i has m_i neighbors, the maximum number of edges that could exist between these nodes equals $\frac{m_i(m_i-1)}{2}$, namely in case each node in the neighborhood is connected to every other node.

Therefore, in an undirected graph with edges E , the local clustering coefficient of a node p_i that has a set of neighbors N_i is defined as

$$\text{local clustering coefficient}(p_i) = \frac{2|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{|N_i|(|N_i| - 1)}.$$

For example, the local clustering coefficient of node C in Figure 3.3.1 is $\frac{2 \cdot 1}{3 \cdot 2} = \frac{1}{3}$, since it has 3 adjacent nodes that could have $\frac{3 \cdot 2}{2}$ interconnecting edges, but there exists only 1 edge between these nodes.

Actor Roles

Actors in software ecosystems are often ascribed certain roles based on their contribution to the ecosystems. Theory about actor roles is published, amongst others, by Kabbedijk & Jansen (2011) (introducing the roles ‘Lone Wolf’, ‘Networker’, ‘One Day Fly’), Manikas & Hansen (2013) (‘Orchestrator’/‘Keystone’, ‘Niche Player’, ‘External Actor’, ‘Vendor’, ‘Customer’), and Eckhardt, Kaats, Jansen, & Alves (2014) (‘Visitor’, ‘Novice’, ‘Regular’, ‘Leader’, ‘Elder’).

Concerning this, we only remark that the term ‘Keystone (Player)’, ‘Orchestrator’, or some other variant is found in many publications. This refers to a central actor in a software ecosystem, responsible for facilitating and sustaining the ecosystem. For example, Apple can be called the keystone of the Apple App Store ecosystem.

3.3.2 Network Level Analysis

The state of an ecosystem as a whole can be assessed using measures from graph theory, similar to those for nodes. Such measures are useful to compare different ecosystems to each other and to measure changes in ecosystems over time. Examples of network properties found in literature are *size*, *robustness*, and *modularity*.

Size

The *size* of a network is defined as the number of nodes in the network. For example, the network in Figure 3.3.1 has size 4.

Network Density

Granovetter (1976) defines the *density* of a network as the number of edges in the network divided by the potential number of edges. In a ‘dense’ network, the number of connections is close to the maximum number of possible connections. In a ‘sparse’ network, the opposite is true.

As we have seen in Section 3.3.1, the potential number of edges in an undirected network with n nodes equals $\frac{n(n-1)}{2}$. Therefore, the network density of a network consisting of n nodes and m edges equals

$$\text{network density} = \frac{2m}{n(n-1)}.$$

For example, the network in Figure 3.3.1 has density $\frac{2 \cdot 4}{4 \cdot 3} = \frac{2}{3}$, which is considered to be relatively dense.

Average Degree

The *average degree* of a network is the average of the degrees of all of its nodes, as defined in the previous section. For example, the average degree of the network in Figure 3.3.1 is $\frac{2+2+3+1}{4} = \frac{1}{2}$.

Average Weighted Degree

The average weighted degree of a network is the average sum of edge weights per node. For example, the average weighted degree of the network in Figure 3.3.1 is $\frac{2+1+2+3+1+3+4+4}{4} = 5$.

Network Degree Centralization

The *centralization* of a network is a measure of how central its most central node is in relation to the other nodes. In a network with a high degree centralization, there will be one (or a few) very central nodes.

Centralization in general is defined by Freeman (1978) as the sum of differences between the centrality of each node and the centrality of the most central node, divided by the maximum possible sum of differences in node centrality. To formalize, let C_i denote the node centrality of node p_i in a network with n nodes. Then

$$\text{network centralization} = \frac{\sum_{i=1}^n C_{\max} - C_i}{\max(\sum_{i=1}^n C_{\max} - C_i)}.$$

There are several types of centralization. In our analysis, we use *degree centralization*, which is based on node degree centrality as defined in Section 3.3.1.

The maximum possible sum of differences in node degree centrality in any network is in case the network has the shape of a star or wheel. In that case, the denominator in the above formula has value $(n-1)(n-2)$ for a network of n nodes (Freeman, 1978). Therefore, the degree centralization of a network with n nodes p_1, \dots, p_n is defined as

$$\text{network degree centralization} = \frac{\sum_{i=1}^n (\text{degree}(p^*) - \text{degree}(p_i))}{(n-2)(n-1)},$$

where p^* is the node with the highest degree in the network.

Average Clustering Coefficient

A network's average clustering coefficient is a measure of the degree to which the nodes in the network tend to cluster together. This measure is defined by Watts & Strogatz (1998) as the average of the local clustering coefficients of all the nodes in the network.

I.e., for a network consisting of n nodes p_1, \dots, p_n ,

$$\text{average clustering coefficient} = \frac{1}{n} \sum_{i=1}^n \text{local clustering coefficient}(p_i).$$

Health

Software ecosystem *health* plays an important role in the literature about software ecosystem analysis. Iansiti & Levien (2004) introduce the characteristics *productivity*, *robustness*, and *niche creation*, which are together used to measure the health of a business ecosystem. These three characteristics are referred to by many other researchers. Among them are den Hartigh, Tol, & Visscher (2006), who make a further distinction between *partner health* and *network health* to assess health of business ecosystems. These measures are applied by others to software ecosystems.

In literature about health of software ecosystems, often the comparison is made with natural health or the health of biological ecosystems, as is done e.g. by van den Berk, Jansen, & Luinenburg (2010), Dhungana et al. (2010) and Mens et al. (2014).

Despite the frequent occurrence in literature of software ecosystem health, the concept often remains implicitly defined, as remarked by Manikas & Hansen (2013):

“Apart from referring to software ecosystem health, very few studies elaborate, analyze or measure the health of a software ecosystem”.

To summarize, while the health of software ecosystems is an intuitive concept that is useful for comparing various ecosystems to each other, few researchers have made an attempt to define this concept as a quantitative measure. As a result, network analysis based on health is still open to interpretation.

3.3.3 Visualization Frameworks and Tools

Scientific frameworks for modeling software ecosystems are given by Boucharas, Jansen, & Brinkkemper (2009), Goeminne & Mens (2010), and Campbell & Ahmed (2010).

A commonly used tool to visualize and analyze Software Ecosystems is *Gephi*, which was introduced by Bastian, Heymann, & Jacomy (2009). An advantage of Gephi is that it can automatically network properties such as Density, Modularity, and Eigenvector Centrality, as e.g. used by Kabbedijk & Jansen (2011). *Gource* (Caudwell, 2010) is a tool that can be used to analyze the history of software components, but it merely shows the evolution of the directory structure of a software repository and its contributors instead of a complete ecosystem and is therefore less useful to visualize or analyze software ecosystems. Other tools for visualizing and analyzing software ecosystems mentioned in literature are the *Software Ecosystem Analysis Dashboard* (Pérez, Deshayes, Goeminne, & Mens, 2012) and the *Small Project Observatory* Lungu, Lanza, Gîrba, & Robbes (2010), both of which do not seem to be publicly accessible.

Visualizations of software ecosystems are often used to analyze relationships and to look for patterns, e.g. clusters in networks or growth of an ecosystem over time. To conclude, visualization of software ecosystems can be helpful for network analysis. There are numerous ways for visualizing ecosystems.

3.4 Evolution of Software Ecosystems

Hanssen (2012) studied an emerging ecosystem for a period of approximately five years and found five major changes in the evolution of that ecosystem, which are to some extent generalizable:

- 1) starting active collaboration with customers and third parties,
- 2) making strategy and plans visible externally,
- 3) opening the technical interface of the product line,
- 4) considering both customers and value-adding third parties as external stakeholders, and
- 5) actively supporting and assisting the community of third parties.

Other longitudinal studies of software ecosystems often describe ecosystem *life cycles*, consisting of several phases or stages in the evolution of ecosystems, comparable to the human phases of birth, adolescence, maturity, and death (Birou, Fawcett, & Magnan, 1997).

3.4.1 Life Cycles

The literature on software ecosystem life cycles finds its roots in the literature on Product Life Cycles, which dates back to the 1960s. Levitt (1965) distinguished four phases in the life cycles of products: Introduction (or Market Development), Growth, Maturity, and Decline. According to this theory, the volume of product

sales typically follows a curve along these phases that increases slowly but exponentially at the beginning, whereafter it stabilizes, and eventually declines. See Figure 3.4.1.

Similarly, dos Santos & Werner (2011) make the comparison to natural ecosystems and suggest the phases of software ecosystem Birth, Development, Maturation, and eventually Death or Transformation. Kim, Lee, & Altmann (2014) mention three consecutive phases: Ascent/Emergence, Maturity/Prosperity, and Decline. dos Santos, Esteves, Freitas, & de Souza (2014) give a curve similar to the Product Life Cycle curve, with stages Initiation, Propagation, Amplification, and Termination, as shown in Figure 3.4.2.

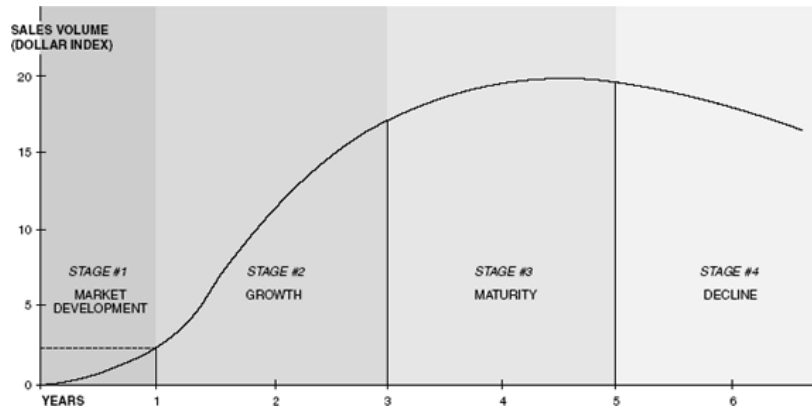


Figure 3.4.1: The Product Life Cycle curve as introduced by Levitt (1965).

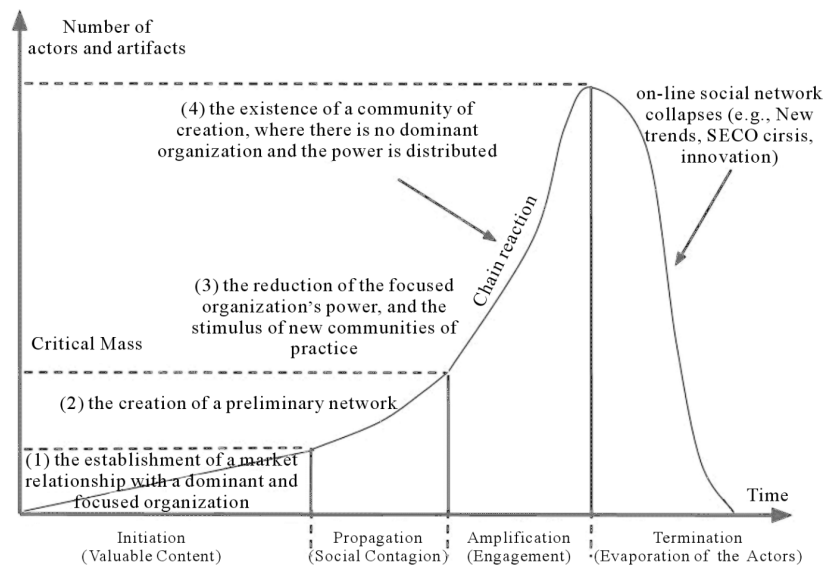


Figure 3.4.2: The software ecosystem maturity curve proposed by dos Santos et al. (2014).

3.4.1.1 What to measure

Product Life Cycle curves typically express the number of sales (sales volume) over time. For software ecosystem life cycles, multiple network properties are suitable to measure. For example, the curve of dos Santos

et al. (2014) describes the “number of actors and artifacts” over time, which is a very general descriptive. In their study, Kim, Lee, & Altmann (2014) present graphs of the centrality of multiple actors in a software ecosystem over a period of time. Several forms of centrality are investigated: degree centrality, eigenvector centrality, and betweenness centrality. Thus, they present a ‘life cycle’ graph for the centrality of nodes in the ecosystem. Similarly, one could make a life cycle graph of a software ecosystem as a whole, based on the evolution of its centralization or some other network property. However, just as the number of sales is the most natural and relevant measure of success of a product, for software ecosystems this is the size (number of nodes) of the ecosystem.

In short, we can say that literature about life cycles of software ecosystems is mostly based on other literature and needs to be validated by quantitative research. The quantity used for obtaining life cycle graphs can be based on several network properties, of which size is the most natural.

3.5 Mining Software Repositories

Definition

Although the term *software repository* is regularly used as referring to “a storage location from which software packages may be retrieved and installed on a computer” (Tao, 2013), in scientific literature the term is often used in a broader sense. Kagdi, Collard, & Maletic (2007) define software repositories as “artifacts that are produced and archived during software evolution”. Software repositories are particularly useful for research, since they “hold a wealth of information and provide a unique view of the actual evolutionary path taken to realize a software system” (Kagdi, Collard, & Maletic, 2007). This information must be extracted or *mined* from the software repositories, which is a research field in itself.

The term *mining software repositories* (MSR) refers to “a broad class of investigations into the examination of software repositories” (Kagdi et al., 2007). The research field of MSR “analyzes and cross-links the rich data available in software repositories to uncover interesting and actionable information about software systems and projects” (Hassan, 2008). In short, the MSR field aims to uncover scientifically relevant information that is stored in software repositories. In recent years, the MSR field has grown to an independent research field, featuring numerous publications and its own international conference¹, of which the 13th edition is to be held in 2016.

Purpose

MSR is applied to get insight into Change Patterns, Defect Analysis, Process and Community Analysis, and Software Reuse (Hassan, Holt, & Mockus, 2004), and more recently also Analysis of Software Ecosystems, Prediction of Future Software Qualities, and Software Project Evolution².

According to Kagdi et al. (2007), a software repository normally contains three basic categories of information that can be mined: software versions, differences between versions, and metadata about the software change (such as author information and information about the context of the change). A researcher should establish

¹<http://msrconf.org>

²See <http://2016.msrconf.org/>

for himself the purpose of mining software repositories, i.e. which category of information he wants to extract. Depending on the software repository and the purpose, a mining method should be established before commencing the mining process.

Thus, we see that MSR is not an end in itself, but should be applied after the researcher has established which information should be obtained by the mining process.

Sources

Early MSR research based on source code often used CVS³ files from SourceForge⁴ as a data source and MSR research based on metadata often extracted data from mailing lists and BugZilla⁵ (Hassan et al., 2004; Kagdi et al., 2007). Nowadays, much MSR research uses GitHub⁶ as a data source (Gousios & Spinellis, 2012; Kalliamvakou, Gousios, Blincoe, Singer, German, & Damian, 2014). GitHub, which hosts both source code and metadata such as bug tracking and feature requests, is currently the largest open source code host in the world (Gousios et al., 2014), hosting more than 25 million software repositories⁷. Hassan (2008) suggests that also *run-time repositories* (e.g. deployment logs of software systems) can be mined to gather useful information.

To summarize, there are different sources for MSR, of which GitHub is gaining in popularity.

Advantages and disadvantages of GitHub as data source

GitHub uses the Git version control system, which is a *decentralized* source code management system. This means that there is no single central repository for projects using Git version control. As opposed to *centralized* source code management systems (such as CVS and Subversion) which have a server-client structure, Git has a peer-to-peer repository structure (Kalliamvakou et al., 2014).

Advantages of a decentralized version control system are that software projects are easy to clone and merge and that changes can be traced back in details. Disadvantages of this are that the history of a software project is not linear and can be more complex than in a centralized version control system (Bird, Rigby, Barr, Hamilton, German, & Devanbu, 2009). However, for projects using GitHub as repository, the repository on GitHub often functions as the origin and as central repository.

Another peril of mining GitHub is that on GitHub, a repository is not necessarily a (software) project. GitHub is also used for e.g. collaboration and version control of T_EX documents⁸.

Most repositories on GitHub have few commits. 90% of the repositories have less than 50 commits, with an average of only 6 commits per repository (Kalliamvakou et al., 2014).

Because forking is easy, a large majority of the repositories are personal repositories. In their research, Kalliamvakou et al. (2014) found that 72% of the repositories on GitHub were personal repositories and that only 54% of all projects were active in the 6 months prior to their research.

³Concurrent Versions System, a version control system.

⁴<http://sourceforge.net/>

⁵<https://www.bugzilla.org/>

⁶<https://github.com/>

⁷See <https://github.com/about/press>

⁸See <http://github.info/>

In conclusion, we can say that, although GitHub is a very valuable and popular data source for MSR research, researchers should be aware of its disadvantages and avoid skewed or biased results.

Mining GitHub

Data from GitHub can be publicly accessed via the GitHub API⁹. Although this data is useful for MSR research, the data that can be retrieved in this way gives only a snapshot for the point in time when it is accessed. However, the GitHub API can be used to obtain a continuous flow of events¹⁰. This way, public events such as the creation of open source repositories, push events, or pull request events can be monitored. However, for all repositories together there are thousands of such events per hour, which is infeasible to keep track of.

The GitHub Archive¹¹ is a project that constantly monitors these events and stores them in JSON format available for download. The project does so since February 2011. Thus, the GitHub Archive project provides valuable longitudinal data for researchers. The advantage of this data is that changes in repositories and users can be studied over a period of time, without having to monitor this data for such a period. The data from the GitHub Archive is mirrored on Google BigQuery¹², making it queryable for researchers without requiring them to download the entire database. However, this service is subject to usage limitations.

The GHTorrent project¹³ (Gousios & Spinellis, 2012) is an effort to create a scalable, queryable, offline mirror of data offered through the GitHub API. The project collects data since mid 2012. The project website claims that data is collected both real-time and backwards, meaning that in the future earlier data might be available through GHTorrent. The content can be queried online using MySQL or MongoDB queries.

'Lean GHTorrent' is an effort to allow researchers to get a slice of the full GHTorrent dataset on demand¹⁴ (Gousios, Vasilescu, Serebrenik, & Zaidman, 2014). However, this work seems to be offline at the time of writing.

To conclude, we see that GitHub provides information about events on their platform in a very transparent way. Such information is valuable for research. Third party projects such as the GitHub Archive and GHTorrent help to digest the data and obtain useful information from it, without requiring the researcher to have significant processing power and storage space available.

⁹<https://developer.github.com/>

¹⁰<https://developer.github.com/v3/activity/events/>

¹¹<https://www.githubarchive.org/>

¹²<https://cloud.google.com/bigquery/>

¹³<http://ghtorrent.org/>

¹⁴<http://ghtorrent.org/lean.html>

Chapter 4

Mining GitHub Data for Collaboration Networks

We limit our research to open source software ecosystems. A key characteristic of open source software is that it is generally co-authored by multiple developers. Therefore we look at co-authorship relationships, which we analyze both between actors and between software components.

4.1 Collaboration Network Identification and Measurement

Co-authorship in Git(Hub) is reflected in the form of ‘push’ and ‘pull request’ events. A push event consists of one or more commits (modifications) that a developer makes to a GitHub repository he is authorized to modify. A pull request is similar to a push event, except that a developer can do this to a repository he is not authorized to modify. First, the user makes a branch or fork (a copy) of original repository. After this copy is modified by the user, he can request the changes to be pulled (copied) to the original repository. An administrator of that repository can then review the pull request and either accept or reject it.

We limit our research to push events and accepted pull requests, since those are most relevant for co-creation. For our research, we utilize these events as follows:

When two users both commit to repository X , we say that there is a relationship between these users. The strength of this relationship depends on how many different repositories these users have recently collaborated on.

Similarly, when a user commits to both repository A and repository B , we say that there is a relationship between these repositories. The strength of this relationship depends on how many different users have recently committed to both A and B .

Nodes

We analyze collaboration networks in the form of undirected weighted graphs. Networks in which each node represents a user we call *user-centered* networks. In *repository-centered* networks, the nodes represent

$$\text{edge weight}(A, B) = \sum_{\text{all users}} \text{recency}(\text{last pull request of this user to A}) \cdot \text{recency}(\text{last pull request of this user to B}).$$

For edges in user-centered graphs, we use an analogous formula.

Example:

Consider a sample set with users X and Y and repositories A and B . Suppose

- user X made his last push (or successful pull request) to repository A 1 day ago and to repository B 2 days ago, and
- user Y made his last push or pull request to repository A 5 days ago and to repository B 3 days ago.

Then

$$\begin{aligned} \text{edge weight}(A, B) &= \left(1 - \frac{1}{30}\right) \cdot \left(1 - \frac{2}{30}\right) + \left(1 - \frac{5}{30}\right) \cdot \left(1 - \frac{3}{30}\right) \\ &\approx 3.32. \end{aligned}$$

Network

We aim to reconstruct networks around a given repository or user, which we call the *origin node*. I.e., starting from the origin node, we look for the software ecosystem of which it is part. Using our definition of edge weight, we can find nodes connected to the origin node. From there, we can find nodes connected to these nodes. We call the number of times this procedure is repeated the *depth* of the network.

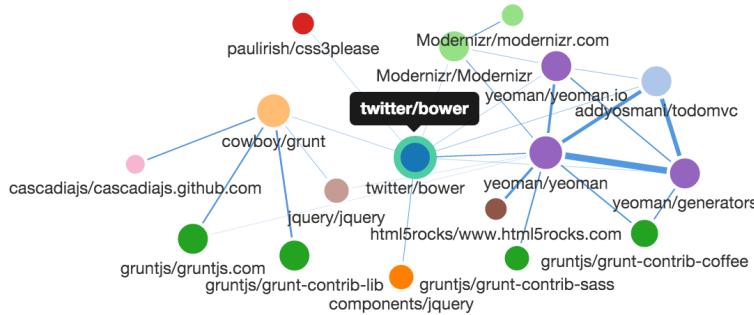


Figure 4.1.2: Repository-centered collaboration network with depth 2 around the GitHub repository `twitter/bower` on October 9th, 2012.

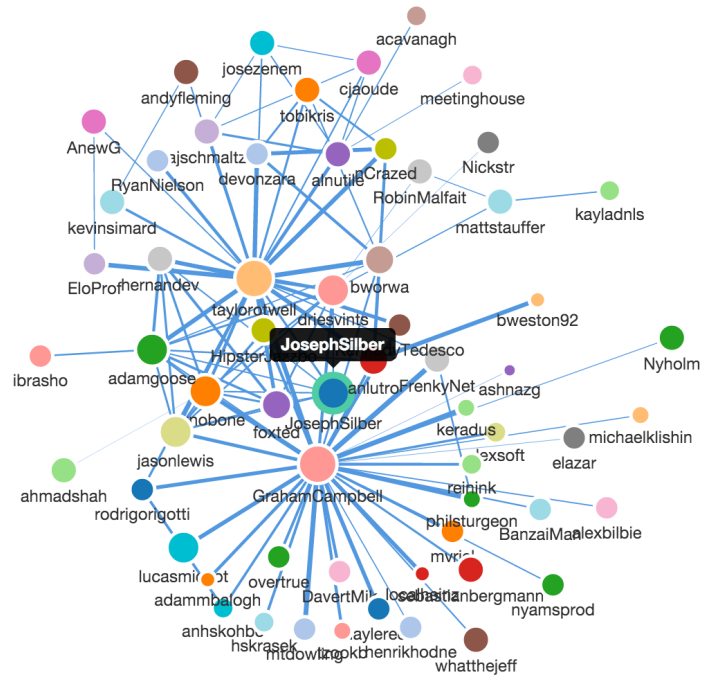


Figure 4.1.3: User-centered collaboration network with depth 3 around the GitHub user JosephSilber on September 19th, 2014.

4.2 Data Collection

The data for our research is obtained from the GitHub Archive database, which contains historical public data from February 2011 until now. For the purpose of our research, we limit our scope of GitHub data to the years 2012, 2013 and 2014, three full years.

We choose to mine our data from the GitHub Archive rather than from GHTorrent since the GHTorrent data (currently) does not go back in history far enough (March 2012). Moreover, since we have to execute complex queries on the data set, we have to download the entire history of push events and pull requests during our scope period, for which we would have to download a 120+ GB database if using GHTorrent¹. The same data can be downloaded in much smaller chunks from the GitHub Archive, from which the relevant data can be extracted, which is only 8.4 GB large when saved using MySQL in InnoDB compact row format².

Issues that arose during the mining process were:

- Data from the GitHub Archive project is stored in JSON³ format, compressed using gzip⁴ and combined as one file per hour. These files are sometimes large to extract in-memory.
- The data files in some cases contain multiple JSON objects per line, which is slightly incorrect JSON and needs to be corrected before further processing.

¹<http://ghtorrent.org/downloads.html>

²See <https://dev.mysql.com/doc/refman/5.7/en/innoDB-row-format.html>

³<http://www.json.org/>

⁴<http://www.gzip.org/>

- Timezones of dates differ per month (some are in UTC, others in PST with daylight saving in summer time), which has to be taken into account.
- During the scope period, the event structure changed a few times, because of introductions of new GitHub API versions.
- The useful mined data has to be stored in a local database, at the end of the mining process containing over 130 million rows of data.

All issues could be taken care of, resulting in an efficiently stored database table that functioned as a further basis for analysis. The 130 million rows of data were stored in an InnoDB table with a size of 8.4 GiB excluding indices.

4.3 Data Processing

The database query to identify ecosystem relationships from the dataset as described in Section 4.1 is relatively complex, especially since it identifies relationships for a period of time. In order to be able to execute this query efficiently, database indices help to reduce execution time and processing power. The trade-off is that extra storage space is required. In our case, an additional 8.5 GB of space was used for database indices.

The algorithm to identify ecosystem relationships as described above has complexity $O(t \cdot n^{d+1})$, where t is the number of days, n is the size of the data set, and d is the depth of the network we want to obtain. This algorithm is in general too complex to execute in real-time, therefore resulting ecosystems are stored in a database as well. Since a network of depth 2 contains the network of depth 1, the additional nodes and edges computed for higher degrees are stored in complement tables.

The code to collect the data and analyze it is stored as an open-source GitHub repository itself⁵.

4.4 Data Analysis

From the resulting networks, properties such as defined in Sections 3.3.1 and 3.3.2 can be measured for each moment in time. These network properties can be displayed as graphs.

As discussed in Section 3.4, size is the most suitable network property to observe as describing ecosystem life cycles.

⁵<https://github.com/jos-/software-ecosystems>

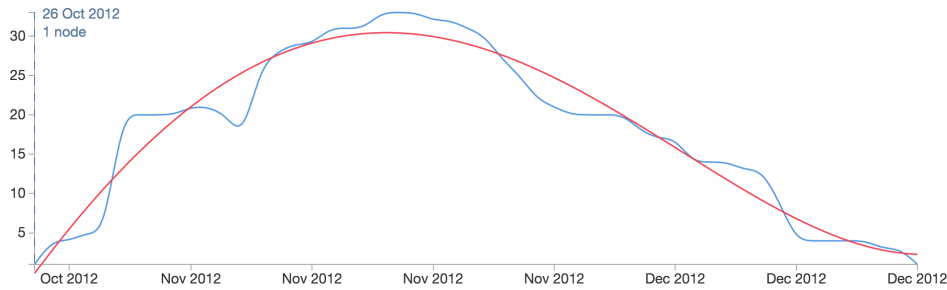


Figure 4.4.1: Size of the user-centered collaboration network with depth 3 around the GitHub user `oszczep`. The blue line indicates the ecosystem size on a day-to-day basis, of which the red line is the fourth order regression polynomial (discussed in the next chapter).

4.5 Results

The tool developed for our analysis can be viewed online⁶. Using our method, software ecosystems can be identified in an objective way. The resulting collaboration networks can be analyzed in detail.

The computation is not instant, but can be regarded to be relatively fast, taking into consideration the size of the data set. However, the complexity of the operation is an issue when trying to identify networks with a high depth. Computation time can be reduced by using indices, however requiring extra storage space.

Our second research question was “How can collaboration in software ecosystems be measured and visualized, based on data from a public data source?”. Our model provides an acceptable method to do so, which is applicable to multiple definitions of software ecosystems. Key features of our model are:

- Based on co-authorship.
- Objective, no human input required, making it possible to analyze large numbers of collaboration networks.
- Suitable for longitudinal analysis.
- Balanced, keeping track of co-authorship for a sliding scale of time.
- Relatively efficient.
- Storing data in a compact way by using complement tables.
- Aimed at GitHub, but suitable to extend to other software repository systems.
- Requiring relatively much storage space, mainly caused by the large number of events that take place on GitHub.

⁶<http://thesis.josvandermaas.nl/>

Chapter 5

Categories of Open Source Collaboration Networks

5.1 Procedure

Our purpose of this chapter is to identify typical categories of open source collaboration networks, based on the life cycles of these networks.

Full life cycles

For our research, we investigate collaboration networks that completed a full life cycle during the period between January 2012 and December 2014. We regard a network to be abandoned when it has no more activity for a period of at least three months. Here ‘no activity’ means that the network has size 1, only containing the origin node, i.e. no relationships between the origin node and any other node for three months.

Thus, we look for collaboration networks on GitHub that had no activity in the first three months and the last three months of our analysis period, but which do have activity in at least some of the other months. Moreover, during their life span, the networks should not have activity gaps (periods of no activity) longer than three months. This way, the life cycles of these networks can be regarded to be complete.

Research samples

We select a random sample of GitHub repositories and another random sample of GitHub users using MySQL’s `ORDER BY RAND()` predicate. This results in a reasonably uniform distribution¹. For our samples, we require a 80% confidence interval and a 6% margin of error.

On December 15th, 2015, GitHub hosted a total number of 30.7 million repositories and 12.1 million users². These numbers, in combination with our confidence interval and margin of error, require sample sizes of at

¹<https://dev.mysql.com/doc/refman/5.7/en/order-by-optimization.html>

²<https://github.com/about/press>

least 114 repositories and another of 114 users (Jones, 1955). From these samples, 114 repository-centered networks and 114 user-centered networks can be identified.

Network depth – Three degrees of separation

We analyze all collaboration networks with a network depth of 3. This means that networks will be identified up to three ‘degrees of separation’ from their root nodes. Thus, we look at the root node, the nodes connected to this node, the nodes connected to those nodes, and finally the nodes connected to those nodes. For example, a path in such an ecosystem can be `twitter/bower – Modernizr/Modernizr – mxcl/homebrew – rails/rails`.

A reason for this is that a higher depth would give a distorted picture, since too distant nodes would be regarded as part of a network. Collins & Chow (1998) suggested that all living humans are interconnected by at most six ‘degrees of separation’. Raising the network depth to 6 would likely result in the same network over and over again, regardless of the root node. Using a network depth lower than 3 would give a less complete picture. Also, network properties such as clustering (which requires a minimum depth of 2 to compute) would be less meaningful for lower degrees.

Another reason for this choice is that researchers found that many forms of human social behavior have an influence up till three degrees of separation. This applies to happiness (Fowler & Christakis, 2008), loneliness (Cacioppo, Fowler, & Christakis, 2009), smoking (J. H. Christakis, 2008), alcohol consumption (Rosenquist, Murabito, Fowler, & Christakis, 2010), depression (Rosenquist, Fowler, & Christakis, 2011), obesity (a Christakis & Fowler, 2007), and, most relevant, cooperative behavior (Fowler & Christakis, 2010). These findings have led to the ‘Three Degrees of Influence’ theory of Christakis & Fowler: “Our influence gradually dissipates and ceases to have a noticeable effect on people beyond the social frontier that lies at three degrees of separation” (Christakis & Fowler, 2009). Despite some critique, this theory has been accepted by many scholars. Since collaboration can be regarded as social behavior and even as part of ‘cooperative behavior’, it seems appropriate to analyze collaboration networks up till three degrees of separation.

Polynomial regression

We study the life cycles of these networks by analyzing the size (number of nodes) of each network over time. This way, we obtain graphs with time on the x-axis and network size on the y-axis. In order to compare these life cycles to each other, we make the data uniform by mapping each graph to the plane $[0, 1] \times [0, 1]$. Thus, time is an interval from 0 to 1 (first and last date of life cycle) and the network size can take all values between 0 and 1 (0 being the minimum size and 1 being the maximum size reached during the life cycle). From this data, we calculate regression polynomials of network size as a function of time.

We use fourth order polynomial regression, so each life cycle is approximated by a function

$$f(x) = a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4,$$

where a_0, \dots, a_4 are constants. These are called the *coefficients* of the polynomial. We also look at lower order regression polynomials, as explained in the next sub-section. We calculate the regression polynomials using the so-called least-squares method (Sorenson, 1970).

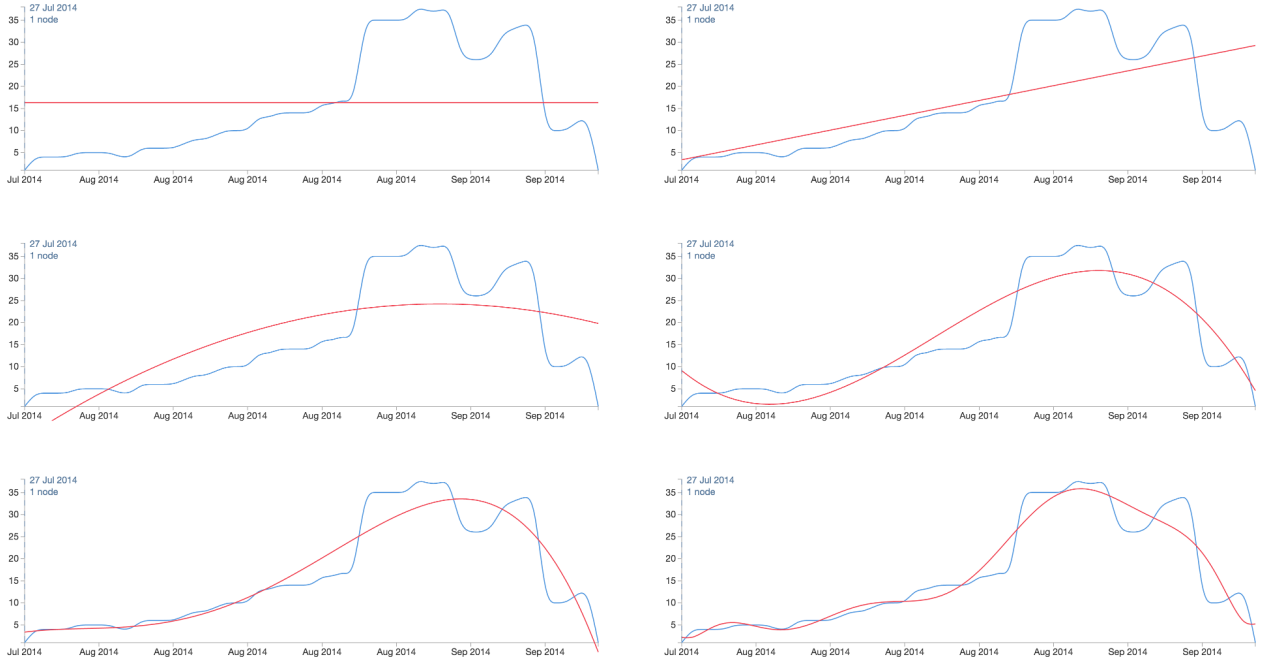


Figure 5.1.1: Regression of the life cycle of repository-centered network with depth 3 around the GitHub user `NSLS-II/userpackages`. The blue line indicates the exact network size per day. The red lines are the regression polynomials of orders 0, 1, 2, 3, 4, and 10, respectively (left to right, top to bottom).

There are several reasons for using fourth order regression instead of a higher or lower order. A lower level would result in worse approximations. As can be seen in Figure 5.1.1, lower order regression allows for less freedom in possible shapes, resulting in a larger possible difference between the original data and the regression line. The least-squares method ensures that the total difference is minimized, but large differences can occur locally when using low order regression. On the other hand, although a regression order higher than 4 would be more accurate, it results in polynomials that are more difficult to compare to each other. The increase in accuracy can be regarded relatively small compared to this drawback. E.g. compare the fourth and tenth order regression lines in Figure 5.1.1.

Categorization

For our research, we want to objectively categorize the collaboration networks based on their life cycles, in such a way that networks with a comparable life cycle shape fall into the same category. To do so, we compare the coefficients of the network size regression polynomials of order 0 to 4. Specifically, we look at the first coefficient of each regression polynomial, i.e.

c_0	in zero order regression polynomial	$f_0(x) = c_0,$
c_1	in first order regression polynomial	$f_1(x) = c_1x + a_0$
c_2	in second order regression polynomial	$f_2(x) = c_2x^2 + a_1x + a_2$
c_3	in third order regression polynomial	$f_3(x) = c_3x^3 + a_3x^2 + a_4x + a_5$
c_4	in fourth order regression polynomial	$f_4(x) = c_4x^4 + a_6x^3 + a_7x^2 + a_8x + a_9.$

The reason for this is that the first coefficient of a regression polynomial contains the most new information when the lower order polynomials are known. E.g. when c_0 is already known, c_1 tells more about the shape of the actual data than a_0 .

Once c_0, \dots, c_4 are known, we use a + sign to indicate that a coefficient belongs to the 50% highest coefficients of that order and a - sign to indicate that it is one of the 50% lowest coefficients of that order. This way, we can classify a life cycle as a combination of + and - signs, e.g. $+-+-$ (meaning $c_0, c_2,$ and c_4 are relatively high and c_1 and c_3 are relatively low). A similar approach is used by Krauthl & Lienert (1978).

The advantage of this approach is that the life cycles are compared on the basis of multiple regression orders and for all of these orders, the life cycles have to be similar to be placed in the same category. To give an indication that the life cycles in a category indeed resemble each other, an overview of the user-centered life cycles belonging to category $+-+-$ is shown in Figure 5.1.2.

Since we look at 5 coefficients per life cycle, we have $2^5 = 32$ possible categories of life cycles, and the probability for a coefficient to be '+' is equal to be '-'. Thus, the statistical probability for a given category to occur is $\frac{1}{32} = 3.125\%$. We analyze our sample sets to see if some categories have a significantly higher rate of occurrence.

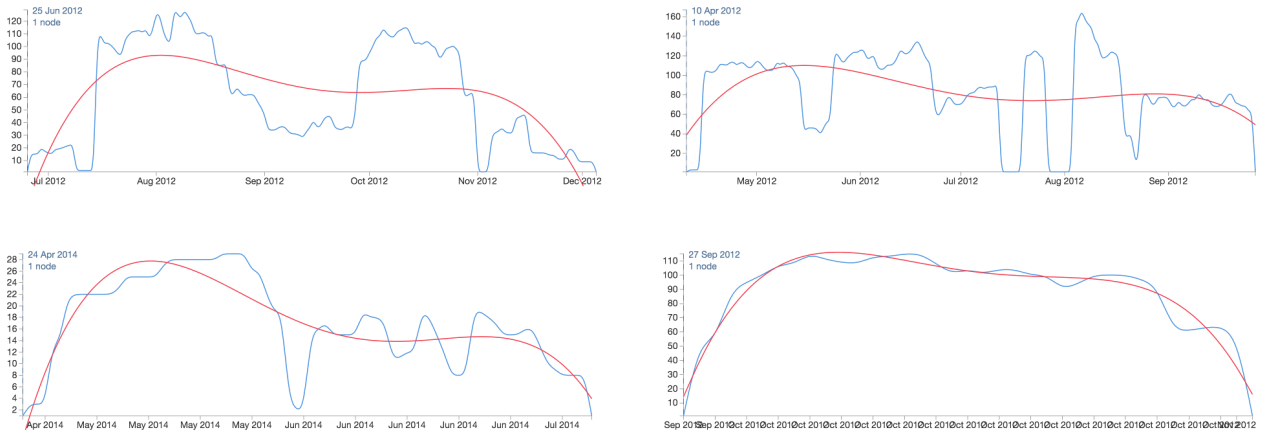


Figure 5.1.2: Life cycles of user-centered networks in the category $+-+-$. Networks in a category have comparable life cycle shapes. Shown from left to right, top to bottom are the life cycles of networks with depth 3 around the users `cgmartin`, `vbardales`, `vadosant`, and `rufinus`.

Validity

Possible drawbacks of this approach are:

- We study relatively short-lasting ecosystems (that are active for a period of at most two and a half years).
- We allow a 6% margin of error and require a confidence level of only 80%.

In relation to the first possible drawback, Kalliamvakou et al. (2014) found that only approximately 2% of the repositories on GitHub have a longer active period than 2,5 years. So our sample represents the vast majority of ecosystems on GitHub. Moreover, we study ecosystems of different lengths (varying from 0 days to 2,5 years), making the results generalizable.

Regarding second drawback, the meaning of this is that for any value we find (e.g. 10% of the ecosystems is of some category X), this means that we are 80% confident that the actual percentage of occurrence lies in the interval of the found value plus or minus the margin of error (in this example, between 4% and 16%). While this is a relative large interval, it allows us to find significantly often-occurring categories with a relatively small sample set. Reducing the margin of error requires much larger sample sets. E.g. to obtain a 3% margin of error instead of 6%, the samples would have to be four times larger (Jones, 1955).

5.2 Hypotheses

In order to conduct our analysis, we formulate the following hypotheses:

1. There are some life cycle categories to which significantly many collaboration networks belong.
2. On average, the size of a collaboration network is lowest during the first quarter of its life cycle and highest during the third quarter.

The second hypothesis is based on the assumption of the ‘typical’ life cycle shape in literature. This shape, which is often found in literature, is characterized by a small size during the first quarter (‘Introduction phase’) and largest during the third quarter (‘Maturity phase’), as can be seen in Figure 5.2.1.

5.3 Results for User-Centered Collaboration Networks

In this section, we analyze the hypotheses for user-centered networks. The obtained sample of these networks, along with their calculated regression polynomials, is shown in Table 9.1.

Hypothesis 1: Different categories

The results contain 24 different life cycle categories, of which the top five is shown below:

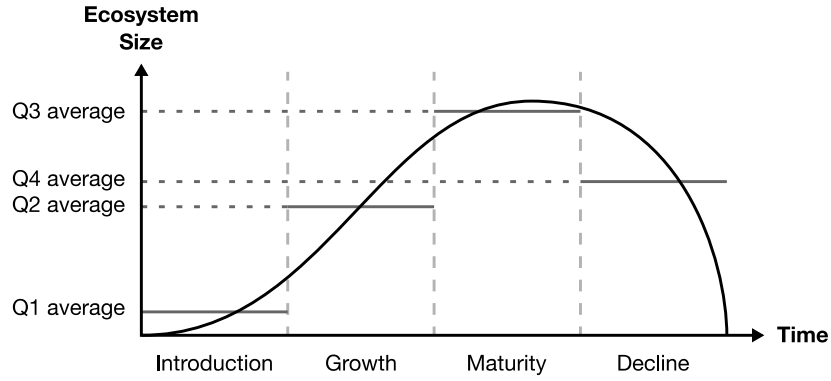


Figure 5.2.1: The typical life cycle found in literature.

Category	Occurrences (out of 114)	Occurrence rate	Significant occurrence	Average fourth order polynomial: $f(x) =$	Shape
--++-	17	14.9%	Yes	$-27.5x^4 + 59.0x^3 - 40.0x^2 + 8.2x + 0.3$	
+---++	11	9.6%	Yes	$-1.4x^4 + 9.3x^3 - 13.2x^2 + 5.4x + 0.3$	
++---	11	9.6%	Yes	$-18.4x^4 + 35.9x^3 - 23.3x^2 + 5.9x + 0.3$	
-++--	9	7.9%	No	$-33.4x^4 + 63.5x^3 - 36.2x^2 + 6.3x$	
-+---+	7	6.1%	No	$7.2x^4 - 17.4x^3 + 12.3x^2 - 2.3x + 0.2$	

We observe that significantly many user-centered collaboration networks to belong to category --++-, +---++, or ++---, so the first hypothesis is accepted.

Hypothesis 2: Typical shape

As can be seen in Table 9.1, the average fourth order regression polynomial is given by $f(x) = -11.7x^4 + 24.6x^3 - 16.9x^2 + 4.0x + 0.3$. Remember that this polynomial describes the data mapped to the plane

$[0, 1] \times [0, 1]$. Thus, the average network size during quarter 1 equals

$$\frac{\int_0^{0.25} (-11.7x^4 + 24.6x^3 - 16.9x^2 + 4.0x + 0.3) dx}{0.25} \approx 0.53.$$

All quarterly averages are given in the table below. Based on this information, the hypothesis is rejected. When taking the average for all user-centered ecosystems, the opposite is true: the network size during is highest during the first quarter and lowest during the third quarter. However, the differences are relatively small.

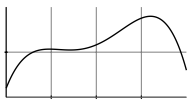
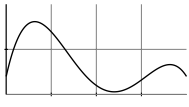

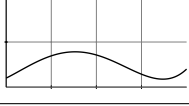
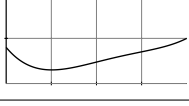
	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Average network size (between 0 and 1)	0.53	0.49	0.43	0.45

5.4 Results for Repository-Centered Collaboration Networks

This section describes the testing of the hypotheses for repository-centered networks, of which the data is shown in Table 9.2.

Hypothesis 1: Different categories

The results for this type of networks contain 28 different life cycle categories (out of 32 possible shapes). The top five of these categories is shown in the table below.

Category	Occurrences (out of 114)	Occurrence rate	Significant occurrence	Average fourth order polynomial: $f(x) =$	Shape
++---	11	9.6%	Yes	$-18.7x^4 + 34.8x^3 - 21.0x^2 + 5.1x + 0.1$	
--+-	9	7.9%	No	$-25.8x^4 + 57.3x^3 - 40.4x^2 + 8.9x + 0.2$	
+---+	9	7.9%	No	$-3.5x^4 + 11.2x^3 - 12.8x^2 + 5.0x + 0.3$	
---++	8	7.0%	No	$5.1x^4 - 7.5x^3 + 1.5x^2 + 1.0x + 0.1$	
-++-+	7	6.1%	No	$3.4x^4 - 8.4x^3 + 7.5x^2 - 2.4x + 0.4$	

From the results in the above table, we see that there is only one category of repository-centered networks to which significantly many networks belong, namely ++---. However, this is enough to for the first hypothesis to be accepted.

Hypothesis 2: Typical shape

The average fourth-order regression polynomial for repository-centered collaboration networks is given by $f(x) = -10.3x^4 + 21.0x^3 - 13.9x^2 + 3.2x + 0.3$. The quarterly average network sizes, based on this equation, are shown in the table below.

	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Average network size (between 0 and 1)	0.48	0.45	0.43	0.46

Based on this information, again the hypothesis is rejected and again, rather the opposite is true. The ‘typical shape’ is not applicable to the average network life cycle. However, it can be applicable to some networks.

When comparing user-centered networks to repository-centered networks, we see that both have a category --+- within the top five categories, with a similar life cycle shape. We see more similarities between user-centered and repository-centered networks life cycles. This is an indication that the two types of networks are essentially not very different. This is further analyzed in the next sections.

Chapter 6

Characteristics of Collaboration Network Categories

In the previous chapter, we identified several popular life cycle shapes. Note that the top three user-centered categories are the same as the top three repository-centered categories, except for the order in which they appear. Another difference is that, for repository-centered networks, only the first category is statistically significant. We will further examine these three life cycle shapes. An overview of these shapes is shown in Table 6.1.

We will use several network properties to analyze whether the collaboration networks that have these life cycle shapes differ from the other networks. The network properties we measure are: density, centralization, and clustering. These properties are based on literature and were discussed in Section 3.3.2. A visual overview of the properties is given in Figure 6.0.1.

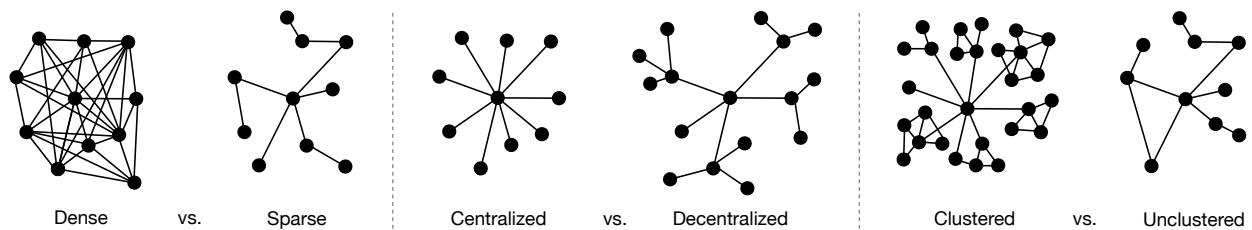


Figure 6.0.1: Network properties: density, centralization, and clustering.

The relevance of these properties for network analysis can be explained as follows.

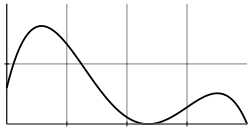
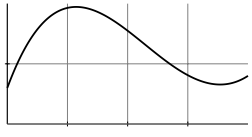
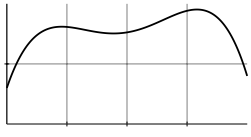
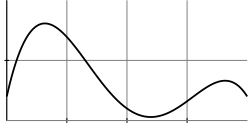
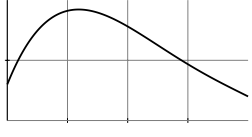
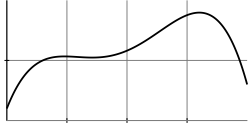
- **Density** is a measure for the number of connections in a network. It indicates how many edges the network has divided by the potential number of edges. Because of its interconnected nature, a dense collaboration network is less likely, or less quickly, to fall apart.
- **Centralization** indicates how central the most central node in the network is in comparison with the other nodes. This tells something about the importance of the most central node (in literature often referred to as the ‘Keystone’) of a network. A high centralization can be beneficial for a network if

the Keystone is a trusted party who maintains the network well, e.g. in case of an App Store. A high centralization can also be a disadvantage for a network in terms of dependency on this node. E.g. when this is a single developer who can stop being active, the collaboration network can quickly fall apart. We measure centralization using Network Degree Centralization, as explained in Section 3.3.2.

- **Clustering** is a measure of the degree to which the nodes in a network tend to cluster together. A high level of clustering is positive in the sense that the network has several sub-networks and is less likely to completely fall apart. However, a high level of clustering may be an indication that the network is fragmented and might split up soon. We measure clustering using the Average Clustering Coefficient, see Section 3.3.2.

For each category, we compare the networks that fall into this category to the networks outside the category. We use the same samples as in the previous chapter. To compare the groups statistically for equality, we conduct Two-sample T-tests. In all tested cases, the samples had variances that had to be regarded unequal (equality of variances was tested using F-tests). Thus, a Welch’s T-test (Welch, 1947) was used to compare the samples for equality. The tests were conducted for both user-centered and repository-centered networks.

Table 6.1: Categories referred to in this section as A, B, and C.

	Category A	Category B	Category C
User-centered equivalent			
Repository-centered equivalent			

Hypotheses

For each of the categories mentioned above (further referred to as categories A, B, and C), we test the following hypotheses:

1. The networks in this category differ significantly from the other ecosystems in terms of network density.
2. The networks in this category differ significantly from the other ecosystems in terms of degree centralization.
3. The networks in this category differ significantly from the other ecosystems in terms of clustering coefficient.

In all cases, the alternative (null) hypotheses assume equality of the groups. Inequality is tested in both directions (smaller and larger), using two-tailed T-tests.

6.1 Category A: Short Revival Before Abandonment

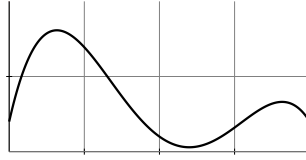


Figure 6.1.1: Life cycle shape of category A.

This type of life cycle is characterized by a quick growth at the start, followed by an early decline. Before the final abandonment of the network, we notice a short revival. In terms of our defined network properties, the networks in this category are characterized by:

- Average density (both for user-centered and repository-centered networks). Therefore, we reject hypothesis 1 for this category. In terms of density, there are no significant differences between this category and the other networks.
- Low centralization (both for user-centered and repository-centered networks). Thus, hypothesis 2 is accepted for this category.
- Low clustering (both for user-centered and repository-centered networks). Hypothesis 3 is accepted for this category.

We conclude that this category is characterized by low centralization (i.e. there is no clear central node, or its role is limited) and low clustering (meaning the nodes do not tend to cluster together). The networks in this category therefore have a relatively unstable network structure. This could be seen as an explanation for the quick decline shortly after reaching the top value.

6.2 Category B: Early Maximum

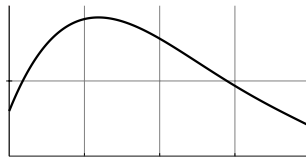


Figure 6.2.1: Life cycle shape of category B.

This life cycle shape is characterized by a relatively quick (although not extreme) growth at the start, followed by a slow but steady decline. Intuitively, this seems like the most natural life cycle for a healthy network. The statistical properties of this category are:

- Average density (both for user-centered and repository-centered networks). Therefore, we reject hypothesis 1 for this category.

- High centralization (significant for repository-centered networks only). We accept hypothesis 2.
- High clustering (significant for user-centered networks only). Hypothesis 3 is accepted for this category.

We see that this category is characterized by a strong network structure. The influence of a central node causes relatively quick growth and the clusters in such a network prevent a quick decline.

6.3 Category C: Extended Growth

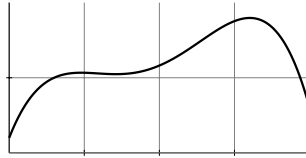


Figure 6.3.1: Life cycle shape of category C.

This life cycle category is distinguished by an extended period of growth, followed by a quick collapse. Moreover, the growth is not uniform, but weakens halfway, after which the growth is continued. This type of life cycle is comparable to the typical product life cycle curve found in literature, although there are some differences. In terms of network properties, this category distinguished itself by:

- Lower than average density (user-centered networks only). Hypothesis 1 is accepted for this category. However, although the difference is large enough to be evaluated as significant, the density of this category is relatively equal to that of the other networks ($p = 4.56\%$).
- Average centralization (both for user-centered and repository-centered networks). Hypothesis 2 is rejected for this category.
- Higher than average clustering (repository-centered networks only). Hypothesis 3 is accepted for this category. Again, the difference is significant ($p = 5.74\%$), but this can be ascribed to our relatively high chosen margin of error.

We conclude that this type of life cycle corresponds to an average network structure in terms of strength. This corresponds with the remark that this life cycle shape is similar to the ‘typical’ (product) life cycle shape. Note that this life cycle category significantly often occurs in both user-centered and repository-centered collaboration networks.

Chapter 7

Collaboration in the Ruby Ecosystem

In the previous chapters, we selected a root node (a user or repository) and from there on identified a collaboration network with a certain depth. Instead of this, we can identify collaboration within a predefined set of users or repositories and construct the collaboration network for this set. We do this for the Ruby ecosystem to analyze how this ecosystem evolved in terms of collaboration.

Ruby

Ruby is a programming language that originated in 1993. Ruby is an object-oriented language, influenced by programming languages Perl, Smalltalk, Eiffel, Ada, and Lisp¹. The first stable release of Ruby was in 1996. Since then, the language has been widely used, being currently listed as the tenth most popular programming language worldwide².

Ruby ecosystem

The Ruby software ecosystem has earlier been analyzed by, amongst others, Kabbedijk & Jansen (2011) and Gregorian (2014). The ecosystem has several characteristics that make it especially interesting for research: Ruby programs and libraries are normally packed in a self-contained format called a ‘gem’, which can be distributed through the website RubyGems.org. Kabbedijk & Jansen (2011) remark that “the entire Ruby ecosystem is a collection of FOSS [Free Open Source Software] projects”. Although this is not necessarily true, the source code of gems be unpacked and viewed, and the large majority of the gems is indeed released under a free license.

Many of the Ruby gems, including the RubyGems library and website itself³, are developed under Git version control and hosted on GitHub. Ruby is currently the sixth most popular programming language among repositories on GitHub, being used in more than 130,000 active repositories⁴.

We analyze the part of the Ruby ecosystem that is hosted on GitHub during the period from January 1st, 2012 till December 31st, 2014. We analyze collaboration within the ecosystem during this period.

¹<http://www.ruby-lang.org/en/about/>

²http://www.tiobe.com/tiobe_index

³<https://github.com/rubygems>

⁴<http://github.info/>

Approach

We identify the Ruby ecosystem on GitHub by selecting all GitHub repositories that have the term ‘ruby’ in their full repository name (e.g. `jruby/perfer` or `mongodb/mongo-ruby-driver`). We measure all collaboration between these repositories in the same way as the repository-centered collaboration networks in the previous chapters were identified. Since we have a predefined set of repositories, we do not need to select a root repository and since we measure all collaboration among the repositories, we essentially obtain a network of infinite depth. Consequently, the nodes in this collaboration network are not necessarily all connected to each other, i.e. we can have separate clusters within this network.

We group repositories per owner, so e.g. `rubyspec/consensus` and `rubyspec/signatures` will be shown together as one node `rubyspec`. Also, we no longer require that at least two users have to collaborate on repositories in order to consider this as a relationship between the repositories. Instead, we limit our analysis to the most important nodes in the Ruby ecosystem, by only selecting nodes that have at least 100 push or pull request events per year on average. This is done to exclude insignificant repositories from the analysis, such as private forks with only a few commits. Many of such insignificant repositories are hosted on GitHub (Kalliamvakou et al., 2014). Furthermore, we exclude all nodes that are not connected to at least one other node in the network.

7.1 Observations

The resulting collaboration network and its properties can be viewed online⁵. We observe that the size of the collaboration network for the Ruby ecosystem is relatively stable throughout our analysis period (2012 till 2014). The network size is especially low in January 2012 and January 2013 and high in September till October 2014, as displayed in Figure 7.1.1. The network size graph is to some extent comparable to the interest in the search term ‘Ruby’ on YouTube⁶ during the same period, as displayed in Figure 7.1.2.

An overview of the most important changes that occur in the collaboration network is given below.

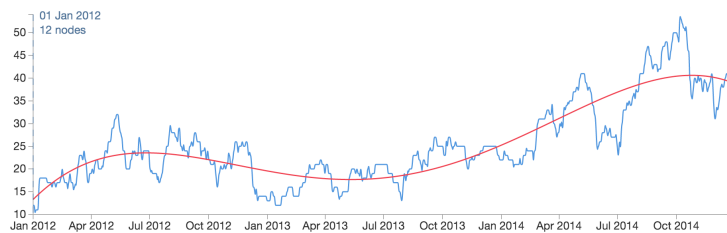


Figure 7.1.1: Network size of the collaboration network for the Ruby ecosystem from 2012 till 2014.

⁵<http://thesis.josvandermaas.nl/ruby>

⁶<https://google.com/trends/explore#q=ruby&gprop=youtube>

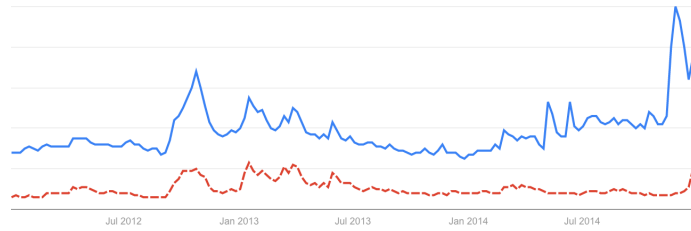


Figure 7.1.2: Popularity on YouTube of the search term ‘Ruby’ (blue) and the part of it that can with certainty be ascribed to the Ruby programming language (red).

Early 2012

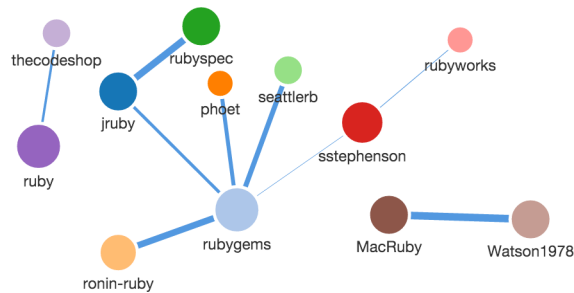


Figure 7.1.3: Collaboration network for the most popular repositories in Ruby ecosystem on January 1st, 2012.

At the start of our observation, `rubygems` is the most central node in the collaboration network. This repository contains the library packaging and distribution system for Ruby that can be used to install or create and distribute Ruby ‘gems’⁷. Consequently, this is one of the most important repositories within the Ruby ecosystem, which is reflected in the collaboration network.

Another important repository is `rubyspec`, an executable specification for the Ruby language⁸. This node is sometimes connected to `rubygems` and later to `mruby`, a lightweight implementation of Ruby with its own interpreter, often used for mobile devices⁹. `mruby` joins GitHub in 2012 and quickly thereafter becomes a central node in the collaboration network.

`ruby` itself¹⁰ (the repository containing the language core) is not very central during this period and is sometimes even excluded from the network as it has no connections to other nodes. `jruby`, an implementation of Ruby on the Java Virtual Machine¹¹, sometimes connects `rubygems` to `ruby` or `rubyspec`.

`MacRuby`, an implementation of Ruby 1.9 directly on top of Mac OS X¹², is relatively solitary, as are `mongodb` (a cross-platform document-oriented database project) and `heroku` (a cloud Platform-as-a-Service). In June, `rubymotion` is introduced, which is a commercial software project. This is an implementation of the Ruby

⁷<https://github.com/rubygems>

⁸See <https://github.com/ruby/rubyspec>, was a separate GitHub until in 2014.

⁹<https://github.com/mruby>

¹⁰<https://github.com/ruby>

¹¹<https://github.com/jruby>

¹²<https://github.com/MacRuby>

programming language that runs on iOS, OS X and Android¹³. `rubymotion` is based on `MacRuby`, having the same main developer¹⁴.

The network size slowly increases early 2012, while density decreases. Centralization follows a high-low-high pattern during this period, but on average decreases. Clustering is low, but increases.

Comments

We observe that nodes that play an important role in the ecosystem have relatively many connections. Visual network analysis can help identify such nodes.

We see the introduction of a commercial gem (`rubymotion`). Although being partly open source, the use of this gem is restricted by a license that is free for “starters and hobbyists”, but demands a fee for commercial use¹⁵. Users that buy a license are given access to additional functionality, which is not open source. The GitHub repository description of the gem explains “This repository contains the parts of the RubyMotion product that are open source. It does not contain the full product, which can be purchased at www.rubymotion.com”. What actually happened here (as can be read on the `rubymotion` project website¹⁶) is that the main developer of both packages first worked for Apple and developed `MacRuby` as a side project. This project became so successful that he decided to leave Apple and start `rubymotion` as a commercial alternative to the popular free gem, being partly open source, but with a license plan for professional use. Later on, we will see that this decision eventually resulted in the abandonment of the open source `MacRuby` project, which is now no longer maintained.

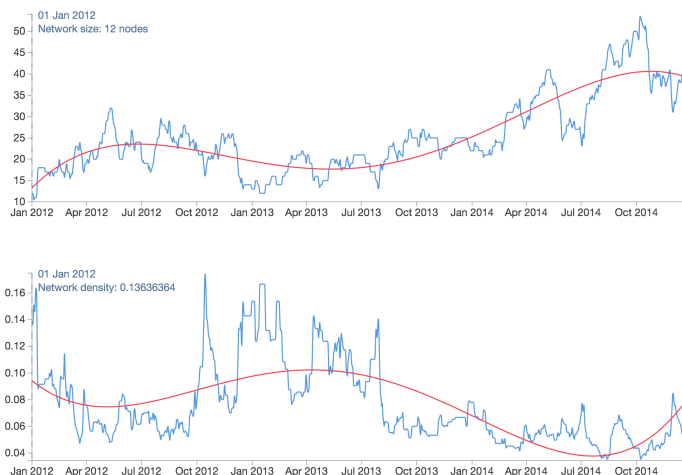


Figure 7.1.4: Size vs. density of the collaboration network of the Ruby ecosystem between 2012 and 2014. These network properties are to some extent inversely proportional.

While the size of the network slowly increases, its density decreases and vice versa. We see this pattern throughout our analysis period, as reflected in Figure 7.1.4. Effectively, this means that whenever new nodes join the network, the average number of connections per node decreases. In other words, newly added nodes

¹³<https://github.com/rubymotion>

¹⁴<http://www.rubymotion.com/about/>

¹⁵See <http://www.rubymotion.com/download/>

¹⁶<http://www.rubymotion.com/about/>

are relatively solitary and nodes that disappear from the network generally have few connections, while more connected nodes remain. On one hand, this is partly trivial behavior. However, the fact that the two measures – size and density – are almost inversely proportional indicates that there is a group of central interconnected nodes that stays in the ecosystem for a long time, while nodes with few connections join and leave the network from time to time.

Late 2012

`rubymotion` becomes more central late 2012. `mruby` becomes more central, but disappears at the end of the period. The network is now characterized by more connections between the nodes, and fewer clusters. A large cluster is formed, consisting of `ruby`, `jruby`, `rubygems`, and `rubyspec`. A small cluster grows, consisting of the two similar implementations of Ruby for Mac OS (`macruby` and `rubymotion`), as earlier discussed. A third, smaller, cluster appears: `postmodern` - `ronin-ruby`. This is a solitary cluster that will stay for quite some time. `postmodern` is a user who created several popular Ruby tools. `ronin-ruby` is a Ruby platform for vulnerability research and exploit development¹⁷.

The network size decreases during this period, especially at the end. Density increases, mainly due to `rubygems` and `jruby`, which become very central nodes during this period. Centralization again follows a high-low-high pattern and further decreases on average. Clustering increases.

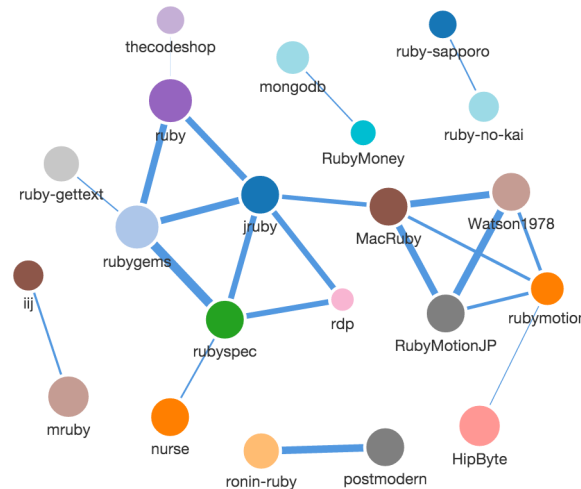


Figure 7.1.5: The Ruby collaboration network after the `ruby` and the `rubymotion` clusters are connected. Network density and clustering have increased.

Comments

We see that the network is divided into several collaboration clusters. A reason for this is that gems that are to some extent similar tend to cluster together, since developers sometimes contribute to multiple comparable gems. An example of two similar gems is `ruby` and `rubygems`, the first one containing the core functionality of the Ruby language and the other being a package manager for Ruby. We see that these two gems are connected during a large part of our analysis period.

¹⁷<http://ronin-ruby.github.io/>

The high-low-high pattern in the clustering of the network, similar to the one between January and June 2012, can be explained as follows. In July, `mruby` is the most central node in the network. However, the centrality of `mruby` decreases while at the same time `rubygems` becomes increasingly central. Halfway this period, `rubygems` has taken over the role of being the most central node in the network. Since network centralization is a measure for how central the network's most central node is in relation to the other nodes, this pattern occurs.

Early 2013

Early 2013 the network size decreases. On February 24th, 2013, ruby version 2.0.0 is released, which gives a boost in network size. The previous stable version, ruby 1.9.3, was released in October 2011. However, the short-time impact on the network size is limited.

The number of small solitary clusters in the network (e.g. `mongodb`'s cluster) increases. The network is divided in two main clusters: `rubyspec` - `jruby` on one hand, `ruby` - `rubygems` on the other. `ruby-no-kai`, a Japanese localization of Ruby, becomes more important. The `postmodern` - `ronin-ruby` cluster grows and joins the `ruby` - `rubygems` cluster.

Density is relatively high during this period, but slightly decreases. Centralization further decreases. This period is characterized by a relatively high level of clustering.

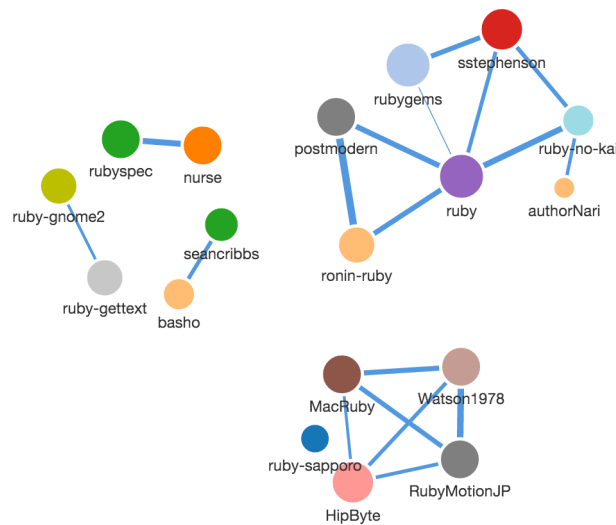


Figure 7.1.6: Early 2013, the collaboration network is characterized by a high level of clustering.

Comments

We observe that the introduction of a new version caused an increase in development activities and collaboration. However, in this case the impact on the network size is lower than expected for such a long-awaited release. A reason for this is that the 2.0.0 version is almost 100% backward compatible with the previous release¹⁸, such that practically no code had to be changed by gem developers in order to migrate to the new

¹⁸https://github.com/ruby/ruby/blob/ruby_2_0_0/NEWS

version. On the long term however, we see a significant growth after this release.

During the period from January till June, clustering is relatively high. The nodes tend to cluster together, resulting in a collaboration network that is divided into several clearly distinguished clusters, as can be seen in Figure 7.1.6.

7.1.1 Late 2013

`jruby` and some time later `rubyspec` disappear from the network. The `postmodern - ronin-ruby` cluster disconnects from the `ruby - rubygems` cluster. The `monkstone - jashkenas` cluster appears, as does the `mongodb - estolfo` cluster. `monkstone` and `jashkenas` are both developers who created a gem to generate computer art using Ruby code. `estolfo` is a user who worked on the documentation of `mongodb`.

Ruby 2.1.0 is released on Christmas Day in 2013, followed by a slow but steady boost in network size. The slowness of the increase in network size can be explained by the fact that the previous stable release was relatively recent.

`MacRuby` disappears in August 2013, never to appear again.

During this period, the network density drops significantly. Centralization further decreases, and reaches its minimum value of our time scope. Clustering significantly drops, but then returns slightly below its old level.

Comments

The commercial gem `rubymotion` was built using the free software `MacRuby` as a basis. We observe that the development of `MacRuby` ends shortly thereafter, resulting in its disappearance from the network in favor of the commercial gem. The free software is currently no longer maintained¹⁹. The core of the `rubymotion` project is contained in the repository `HipByte/rubymotion` (`HipByte` being the company that has the ownership of the software). Interestingly, the repository of this commercial gem has approximately 50 contributors²⁰, most of which contribute to it for free. We see that the decision to make part of this software free for individual use benefits the owner in the sense that the software is co-created by outside developers.

The drop in network density can be explained by the introduction of several two-node clusters, together with a decomposition of existing larger clusters.

7.1.2 Early 2014

The `heroku - hone` cluster appears. More small solitary clusters start to form. `mruby` is reincluded in the network and again quickly becomes a central node.

`ruby` also becomes a central node and attracts some smaller clusters. `ruby - processing` joins the `monkstone - jashkenas` cluster. There are now three main clusters: `ruby - rubygems`, `mruby`, and `rubymotion`.

¹⁹<https://github.com/MacRuby/MacRuby>

²⁰<https://github.com/HipByte/RubyMotion>

Network size continues its increase after last Ruby version release. Density drops further, but centralization finally increases. Clustering drops, but then increases mid 2014.

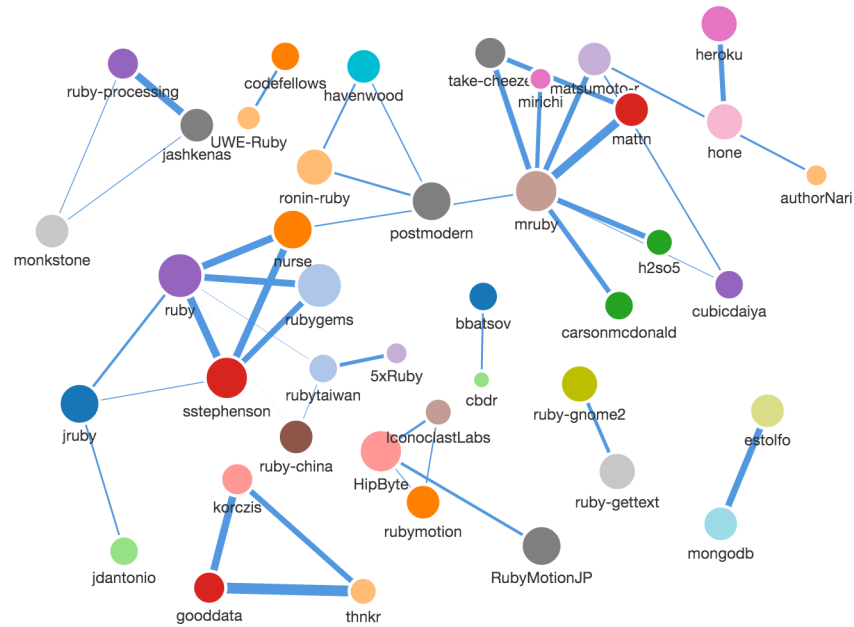


Figure 7.1.7: The Ruby collaboration network in May 2014, after the introduction of many new nodes. Clearly visible is that `ruby` and `mruby` are central nodes.

Late 2014

More two-node clusters are introduced that disappear at the end of the period. In August 2014, `ruby`, `mruby`, and `postmodern` are connected, forming a large cluster. The network is centralized around these nodes and `rubygems`. `jruby` re-appears, forming a cluster on its own. The separate `rubymotion` cluster grows.

At the end of 2014, all main nodes are in some way connected to each other. Network size further increases, especially around October and November. Ruby 2.2.0 is released on Christmas Day in 2014. Since this is a few days before the end of our time scope, it is difficult to tell what the effect of this release is on the network is. Density increases again, and centralization and clustering further increase.

Comments

We have seen the network develop from relatively small to a larger network with a complex structure. At the same time, we observe that some important nodes (e.g. `ruby`, `rubygems`, `rubyspec`, `mruby`, and `rubymotion`) have been included in the network almost the entire time and often take the most central positions. It turns out that collaboration in the Ruby ecosystem mostly takes place around the most important nodes, around which clusters of similar projects are formed.

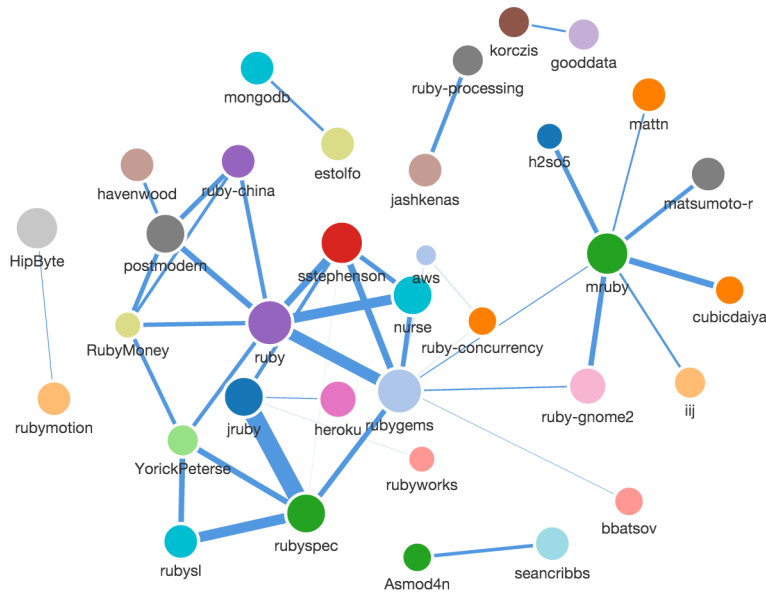


Figure 7.1.8: Ruby ecosystem collaboration network on December 31st, 2012. Most nodes are via some path connected to each other.

7.2 Summary of Observations

A collaboration network gives us information about how collaboration takes place in practice. We observe temporary declines in collaboration activity around the months January - February and July - August in all three analyzed years (see Figure 7.1.1). These deviations can be explained by holiday periods, during which development activity – and as a result also collaboration – is lower than average.

Our analysis period contains two important releases of the core software of the ecosystem (a third one occurring at the very end of our scope). One of these releases has a small impact, whereas the other one is followed by a significant increase in collaboration activity. Here we see that not only the size of a release (the number of changes) and its timing, but also its nature (being backwards compatible or not) is determinative for its impact.

Throughout the analyzed period, the collaboration network turns out to be dynamic and its size volatile, partly due to conventions on which the analysis is based. We observe a more or less permanent basis of central nodes, around which clusters form. In the periphery, many two-node clusters appear. While most of these small clusters turn out to be short-lasting, some of them endure for a long time and succeed to grow to significant positions in the ecosystem.

Density is to a large extent the inverse of network size, confirming the existence of a steady core of nodes. The role of the most central node in the network changes from time to time.

In a mainly open source ecosystem, projects related to commercial software can emerge. Such partly open source software can benefit from outside contributors. However, as observed, commercial software can gain popularity at the cost of free and open source alternatives.

Chapter 8

Discussion

We extracted information about collaboration evolution that is implicitly available in publicly accessible data. In general we can say that more collaboration is beneficial for a software ecosystem. But the exact nature of this collaboration is determinative for its benefit for software ecosystems.

Visual analysis is helpful for the understanding of ecosystems and gives an overview of what happens in terms of collaboration. Our research results are particularly relevant for the following groups of stakeholders:

- Software developers who have to make a choice between various available open source libraries and want to choose the library embedded in the most healthy collaboration network.
- An author or keystone player of an open source software component, to help identify opportunities and threats for its collaboration structure.
- A single developer, for awareness of the position of his software component and the context in which it is embedded.
- Market researchers that want to identify niche markets in open source networks.
- A developer who wants to contribute to an ecosystem, to identify where possibilities lie.
- Researchers to help validate ecosystem maturity models.
- Researchers who want to conduct further network analysis in the field of software ecosystems.

8.1 Findings and Implications

Relationships

Our first research question was:

RQ1: What are relationships in software ecosystems?

First of all, we observed that there are different definitions of what a software ecosystem is. The chosen definition is determinative for the perception and analysis of ecosystems. The most cited definition is that of Jansen et al. (2009), which we also use. However, we remark that software ecosystem relationships can exist both between actors and between software components, as reflected in the other definitions.

Although all definitions mention the existence of relationships in software ecosystems, the exact nature of these relationships is not strictly defined. As a result, we find in literature numerous ways on the basis of which ecosystem relationships are identified and measured.

The essence of software ecosystem relationships is the exchange of information. Through ecosystem relationships, information is exchanged between actors or between software systems. Often, this exchange is facilitated by a shared technology or platform. Ecosystem relationships are dynamic and can in general be measured in terms of continuation and strength.

An implication of this is that the researcher is relatively free to determine a measure for relationships. Many publications start with one well-established software ecosystem as a basis and then measure the relationships within this ecosystem. Relationships are rarely used to identify previously unknown ecosystems.

Since collaboration is a key aspect of open source software, this is a suitable measure for relationships in open source ecosystems. Several scientific sources use collaboration as a basis for ecosystem relationships. Such relationships can be identified and measured by mining software repositories.

Collaboration network modeling method

Our second research question was formulated as follows:

RQ2: How can collaboration in software ecosystems be measured and visualized, based on data from a public data source?

Modeling software ecosystems using a network structure is useful for the understanding and analysis of ecosystems. Nodes in such a network normally represent actors, but can also refer to software components. Edges indicate relationships between the nodes. Once the network is identified, numerous analysis methods from graph theory are available, both on node level and on network level.

Using GitHub as a data source has its limitations, but when the researcher is aware of these, it can be a very suitable data source because of its widespread use and the wealth of information it contains. This information can be transparently accessed through the GitHub API or using external projects, some of which store historical data that is useful for longitudinal analysis.

We use such historical data, which we analyze using a sliding time scale for the measurement of collaboration. As a result, we obtain collaboration networks that can be calculated on a day-to-day basis or even more precisely. This provides insight in the a dynamics of ecosystem evolution.

Our model for identifying collaboration relationships has as a benefit that the analysis can be done in retrospect without influencing the research subjects. Another clear benefit is that it does not require subjective human input or assessment. It does, however require to set a root node and a depth to indicate how far the analysis should reach. When these are provided, the collaboration network around such a node can be identified.

Computation is a challenge because of the large amount of data and the large number of possible relationships that have to be checked. However, this can be done in a relatively efficient way.

Life cycles

Our third research question was:

RQ3: Can various categories of collaboration networks be distinguished, based on their life cycles?

Since our method for measuring collaboration in software ecosystems can be executed in an automated way, it allows us to analyze large numbers of collaboration networks. We chose to select a sample set of 114 so-called user-centered collaboration networks (networks with users as nodes) and another one of 114 repository-centered networks (having repositories as nodes). Although these sample sizes are significantly higher than the average found in existing literature, the samples are still relatively small compared to the large numbers of users and repositories that are active on GitHub. Consequently, we are only 80% confident that our findings correspond with reality and allow for an error margin of 6%.

Our scope is the years 2012 till 2014. Since we want to measure life cycle shapes, we need to ensure that shapes we measure are full life cycles. Therefore, our sample sets consist of collaboration networks of which we are confident to have completed a full life cycle in this period.

The observed life cycles can be volatile, partly because of the choice to analyze collaboration based on the last 30 days prior to a measure point. Regression helps to find a pattern in life cycles and to categorize their shapes.

We observe that there is no standard life cycle shape for collaboration networks in open source ecosystems, but many variations are possible. We categorize these life cycles in such a way that each category has an equal mathematical probability for a life cycle to fit into. However, our analysis points out that some life cycle shapes occur significantly often.

The results are to some extent similar for user-centered collaboration networks and repository-centered networks. We see the comparable shapes occurring in both types of networks. Remarkably, the ‘typical’ life cycle shape assumed in several scientific sources (having a low first quarter and a high third quarter) does not occur significantly often. In general, the decline of a collaboration network’s size takes more time. We observe three clearly distinguished life cycle shapes that occur in our sample set.

Characteristics

After this followed a fourth research question:

RQ4: How are the collaboration networks in these categories characterized?

We analyzed the differences between the collaboration networks having one of the three identified common life cycle shapes to the networks that had a different life cycle shape. This analyzed was based on the network properties density, centralization, and clustering.

Our observation was that different life cycle shapes corresponded to significantly different network structures. Network structure turned out to be determinative for the stability of a network's life cycle. A strong network structure can prevent sudden collapse of a collaboration network.

When a network is distinguished by low centralization and low clustering (a weak network structure), then a quick collapse of the network can be expected, possibly followed by a short-term revival before the final abandonment of the network. On the other hand, when a network is distinguished by high centralization and high clustering (a strong network structure), one can expect a relatively stable period of growth, followed by a slow and more predictable process of decline. When centralization and clustering are average, we can expect a growth that is comparable to the 'typical' life cycle, apart from being slightly steeper at the beginning and having a fluctuation in the middle. This finding matches to some extent with assumptions in existing literature. Network density was not found to have a significant effect on the life cycle shape itself, but could have an effect on the duration of a life cycle and its maximum.

Ruby ecosystem

Our final sub research question was formulated as:

RQ5: How does collaboration in an open source ecosystem evolve in practice?

For analyzing collaboration within a predefined set of nodes, no root node and depth are required to be set. All collaboration among the Ruby repositories on GitHub was identified for the period from 2012 till 2014. This raises a question: which sub-part of this ecosystem's life cycle did we analyze? Since the Ruby ecosystem is still in development, this is difficult to tell. Based on the information that the Ruby ecosystem emerged around 1996 and information from popularity trackers such as Google Trends¹, we can presume that the Ruby ecosystem has started its decline. So in the big picture, we likely analyzed a small part of the decline process of the Ruby ecosystem.

Observing such a collaboration network can help to increase awareness of the position of nodes. The centrality of nodes in the resulting network is an indication of the role they play. In general, more important nodes take a more centralized position in the collaboration network. Certain nodes continue to stay in the collaboration network during the entire observation period.

Collaboration is influenced by external factors (e.g. public holidays), internal factors (e.g. software versions and releases) and of the popularity of software. Changes in an ecosystem's collaboration network can again be measured in terms of density, clustering and centralization.

We observe that the collaboration network for the Ruby ecosystem is characterized by several clusters of software among which collaboration takes place. Sometimes two of such clusters join, while others disconnect and fall apart.

An important remark is that projects in the Ruby ecosystem can be partly open source, being used in a business model to attract developers that have to pay a license when using the software commercially. Some of such commercial projects take the place of free alternatives.

¹<https://google.com/trends/explore#q=%2Fm%2F06ff5>

8.2 Validity

When analyzing collaboration in open source software ecosystems, choices have to be made to create a model of the actual world. Choices made in our approach are:

- Using GitHub as a data source, since it is the largest code host in the world (Gousios, Vasilescu, Serebrenik, & Zaidman, 2014).
- Measuring relationships and relationship strength based on collaboration.
- Only regarding activity that occurred in the past 30 days prior to a measure point.
- Requiring a minimum of two collaborating users or repositories to be included in the analysis, in order to prevent false positives.
- Regarding a period of three full months of no collaboration as an indication that a network is inactive.
- Calculating networks using depth 3, based on scientific sources.
- Using polynomial regression of order 1 till 4 to normalize and analyze data.
- Identifying the Ruby ecosystem on GitHub as all repositories containing ‘ruby’ in their full name.

Although the above choices can all be explained and are supported by scientific evidence, they make the analysis quite specific. We remark again that most findings are under the reservation that we are statistically only 80% confident that our findings correspond with reality. For further discussion of the validity of our research approach, we refer to Section 2.4.

Our findings are limited to the scope of collaboration in open source software ecosystems and as such describe only one aspect of ecosystem evolution. Moreover, the findings cannot be generalized to the entire research field of software ecosystems, but are restricted to open source software.

8.3 Future Research

In our analysis, relatively much time was spent developing a suitable analysis tool. Future research (possibly using the same tool or a similar procedure) could be more in-depth, focusing on one of the following areas:

- Disturbances in life cycles: What happens with life cycles when events such as the creation of branches and forks (copies) take place? What is the effect of such events on collaboration evolution?
- Roles in ecosystems: Which roles do actors and repositories in software ecosystems have? Can certain common roles be distinguished? Are certain roles more important than others, or do they complement each other? What is the effect of a change in roles on an ecosystem?
- More technical network analysis: Once an ecosystem network is identified, available algorithms from graph theory can be applied to e.g. look at route problems, network flow, and node clustering. A substantial number of such algorithms is available, some of which can shed new light on the nature of software ecosystems.

- Ecosystem maturity: More attention to ecosystem maturity and life cycle phases, such as Introduction, Growth, Maturity, and Decline. Can such phases be objectively identified? What are indicators for reaching a higher maturity level?
- More statistical analysis: Following collaboration networks for a longer period of time, using larger samples and using more statistical analysis methods to identify causal relationships.
- Prediction of life cycles: Exploration of patterns found in comparable networks that are ahead in time. Early indicators could possibly give information on how the life cycle of a network will develop.
- Comparison between open and closed source: What are the differences between open source and closed source software ecosystems? To which extent can the results for open source ecosystems be generalized for software ecosystems in general?

Chapter 9

Conclusion

Relationships in software ecosystems represent information exchange. Since various information flows can be identified among software developers and software components, the procedure of relationship measurement is decisive for the perception and further analysis of software ecosystems.

We present a method to objectively identify and measure collaboration networks in open source ecosystems. This method can be executed in automated ways, making it possible to perform statistic analysis on collaboration evolution. We studied 228 collaboration networks that emerged and dissolved in the period from January 2012 till December 2014 and as such followed complete life cycles.

We analyzed the life cycle shapes of these collaboration networks, based on network size. Many different shapes are possible, some of which occur significantly often. The most significant shapes turn out to be slightly different from a standard shape that is often assumed in literature. Next, we analyzed which network structures correspond to the observed life cycle shapes. A strong network structure can benefit a collaboration network in the sense that it is less likely, or less quickly, to fall apart. A network's level of centralization and clustering turn out to be indicators of the stability of the network's life cycle.

Analyzing an existing ecosystem's collaboration network can increase awareness about the position and influence of actors in the ecosystem. Collaboration evolution must be seen in its context, sometime influenced by external factors.

The results of our analysis shed new light on the evolution of collaboration within open source software ecosystems. The followed procedures for collaboration identification and life cycle assessment can benefit scientific research in the field of software ecosystems.

Bibliography

- van Angeren, J. (2013). Exploring Platform Ecosystems: A Comparison of Complementor Networks and their Characteristics. *Master Thesis*.
- van Angeren, J., Alves, C., & Jansen, S. (2014). Analyzing Complementor Interactions in Commercial Platform Ecosystems: The Role of Governance Mechanisms.
- Arksey, H., & O'Malley, L. (2005). Scoping studies: Towards a methodological framework. *International Journal of Social Research Methodology*, 8(1), 19–32.
- Barbosa, O., & Alves, C. (2011). A Systematic Mapping Study on Software Ecosystems, 15–26.
- Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: An Open Source Software for Exploring and Manipulating Networks. *Third International AAAI Conference on Weblogs and Social Media*, 361–362.
- van den Berk, I., Jansen, S., & Luinenburg, L. (2010). Software Ecosystems: A Software Ecosystem Strategy Assessment Model. *European Conference on Software Architecture*, 127–134.
- Bird, C., Gourley, A., Devanbu, P., Gertz, M., & Swaminathan, A. (2006). Mining email social networks, 137–143.
- Bird, C., Rigby, P.C., Barr, E.T., Hamilton, D.J., German, D.M., & Devanbu, P. (2009). The Promises and Perils of Mining Git, 1–10.
- Birou, L., Fawcett, S.E., & Magnan, G.M. (1997). Integrating Product Life Cycle and Purchasing Strategies. *Journal of Supply Chain Management*, 33(1), 23–31.
- Blincoe, K., Harrison, F., & Damian, D. (2015). Ecosystems in GitHub and a Method for Ecosystem Identification using Reference Coupling. *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15), Florence, Italy*.
- Bosch, J. (2009). From software product lines to software ecosystems. *Proceedings of the 13th International Software Product Line Conference*, (Splc), 111–119.
- Boucharas, V., Jansen, S., & Brinkkemper, S. (2009). Formalizing software ecosystem modeling. *Proceedings of the 1st international workshop on Open component ecosystems IWOCE 09*, 19(2), 41.
- Cacioppo, J.T., Fowler, J.H., & Christakis, N.a. (2009). Alone in the crowd: the structure and spread of loneliness in a large social network. *Journal of personality and social psychology*, 97(6), 977–991.

- Campbell, P., & Ahmed, F. (2010). A three-dimensional view of software ecosystems. *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, (c), 81–84.
- Caudwell, A.H. (2010). Gource: visualizing software version control history. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, ACM, (pp. 73–74).
- a Christakis, N., & Fowler, J.H. (2007). The spread of obesity in a large social network over 32 years. *The New England journal of medicine*, 357(4), 370–9.
- Christakis, N.A., & Fowler, J.H. (2009). *Connected: The Surprising Power of Our Social Networks and How They Shape Our Lives*, vol. 3. Little, Brown and Company.
- Collins, J.J., & Chow, C.C. (1998). Network Modelling [Six Degrees of Separation]. *Nature*, 393, 409–410.
- Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday*, 10(2).
- Dhungana, D., Groher, I., Schludermann, E., & Biff, S. (2010). Software ecosystems vs. natural ecosystems. *Ecosystems*, 96–102.
- Eckhardt, E., Kaats, E., Jansen, S., & Alves, C. (2014). The Merits of a Meritocracy in Open Source Software Ecosystems. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ACM, (p. 7).
- Fowler, J.H., & Christakis, N.A. (2008). Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the Framingham Heart Study. *BMJ*, 337(a2338), 1–9.
- Fowler, J.H., & Christakis, N.a. (2010). Cooperative behavior cascades in human social networks. *Proceedings of the National Academy of Sciences of the United States of America*, 107(12), 5334–5338.
- Freeman, L.C. (1978). Centrality in social networks conceptual clarification. 1(3), 215–239.
- Goeminne, M., & Mens, T. (2010). A framework for analysing and visualising open source software ecosystems, 1 – 6.
- Gousios, G., & Spinellis, D. (2012). GHTorrent: Github’s data from a firehose, 12–21.
- Gousios, G., Vasilescu, B., Serebrenik, A., & Zaidman, A. (2014). Lean GHTorrent: GitHub data on demand. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, 384–387.
- Granovetter, M. (1976). Network Sampling: Some First Steps. 81(6), 1287.
- Gregorian, A. (2014). A Tie Strength Model For Reconstructing Collaboration Networks on GitHub – A study of the Ruby on Rails project network. *Master Thesis*.
- Hanssen, G.K. (2012). A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. *Journal of Systems and Software*, 85(7), 1455–1466.

- den Hartigh, E., Tol, M., & Visscher, W. (2006). The health measurement of a business ecosystem. In *Proceedings of the European Network on Chaos and Complexity Research and Management Practice Meeting*, (pp. 1–39).
- Hassan, A.E. (2008). The road ahead for Mining Software Repositories, 48–57.
- Hassan, A.E., Holt, R.C., & Mockus, A. (2004). MSR 2004: International workshop on mining software repositories. 26, 770–771.
- Holmlund, M. (1997). What are relationships in business networks? *Management Decision*, 35(4), 304–309.
- Hoving, R., Slot, G., & Jansen, S. (2013). Python: Characteristics identification of a free open source software ecosystem, 13–18.
- Iansiti, M., & Levien, R. (2004). Strategy as ecology. *Harvard business review*, 82(3), 68–81.
- J. H. Christakis, N. A., .F. (2008). The collective dynamics of smoking in a large social network. *New England journal of medicine*, 358(21), 2249–2258.
- Jansen, S., & Bloemendal, E. (2013). Defining app stores: The role of curated marketplaces in software ecosystems. 150 LNBIP, 195–206.
- Jansen, S., Finkelstein, A., & Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. *2009 31st International Conference on Software Engineering - Companion Volume*.
- Jones, L.V. (1955). Statistical theory and research design. *Annual Review of Psychology*, 6(1), 405–430.
- Kabbedijk, J., & Jansen, S. (2011). Steering insight: An exploration of the Ruby software ecosystem. 80 LNBIP, 44–55.
- Kagdi, H., Collard, M.L., & Maletic, J.I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. 19(2), 77–131.
- Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., & Damian, D. (2014). The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, New York, NY, USA: ACM, (pp. 92–101).
- Kim, K., Altmann, J., & Lee, W.R. (2013). Patterns Of Innovation In SaaS Networks: Trend Analysis Of Node Centralities. In *ECIS*, (p. 187).
- Kim, K., Lee, W.R., & Altmann, J. (2014). SNA-based innovation trend analysis in software service networks. *Electronic Markets*.
- Krauthl, J., & Lienert, G.A. (1978). Nonparametric Two-Sample Comparison of Learning Curves Based on Orthogonal Polynomials. *Psychological Research*, 40, 159–171.
- Levitt, T. (1965). *Exploit the product life cycle*, vol. 43. Graduate School of Business Administration, Harvard University.

- Lungu, M., Lanza, M., Gîrba, T., & Robbes, R. (2010). The Small Project Observatory: Visualizing software ecosystems. *Science of Computer Programming*, 75(4), 264–275.
- Manikas, K., & Hansen, K.M. (2013). Software ecosystems – A systematic literature review. *Journal of Systems and Software*, 86(5), 1294–1306.
- Mens, T., & Goeminne, M. (2011). Analysing the evolution of social aspects of open source software ecosystems. *Proc. 3rd Int. Workshop on Software Ecosystems*, 1–14.
- Mens, T., Claes, M., Grosjean, P., & Serebrenik, A. (2014). Studying Evolving Software Ecosystems based on Ecological Models. In T. Mens, A. Serebrenik, & A. Cleve (Eds.), *Evolving Software Systems*, Springer Berlin Heidelberg, (pp. 297–326).
- Messerschmitt, D.G., & Szyperski, C. (2003). *Software Ecosystem: Understanding an Indispensable Technology and Industry*, vol. 1.
- Miluzzo, E., Lane, N.D., Lu, H., & Campbell, A.T. (2010). Research in the App Store Era: Experiences from the CenceMe App Deployment on the iPhone. *First Workshop on Research in the Large at UbiComp*.
- Molder, J.T., Van Lier, B., & Jansen, S. (2011). Clopenness of systems: The interwoven nature of ecosystems. *Third International Workshop on Software Ecosystems (IWSECO-2011)*, CEUR-WS, 746, 52–64.
- Pérez, J., Deshayes, R., Goeminne, M., & Mens, T. (2012). SECONDA: Software ecosystem analysis dashboard, 527–530.
- Polli, R., & Cook, V. (1969). Validity of the product life cycle. *Journal of Business*, 385–400.
- Rosenquist, J.N., Murabito, J., Fowler, J.H., & Christakis, N.A. (2010). The Spread of Alcohol Consumption Behavior in a Large Social Network. *Annals of Internal Medicine*, 152(7), 426–433.
- Rosenquist, J.N., Fowler, J.H., & Christakis, N.a. (2011). Social network determinants of depression. *Molecular psychiatry*, 16(3), 273–281.
- Santana, F., & Werner, C. (2013). Towards the Analysis of Software Projects Dependencies: An Exploratory Visual Study of Software Ecosystems. . . . *Workshop on Software Ecosystems, 4th . . .*, 1 – 12.
- dos Santos, R.P., & Werner, C. (2011). Treating business dimension in software ecosystems. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, ACM, (pp. 197–201).
- dos Santos, R.P., Esteves, M.G.P., Freitas, G.d.S., & de Souza, J.M. (2014). Using Social Networks to Support Software Ecosystems Comprehension and Evolution. *Social Networking*, 3(2).
- Sorenson, H. (1970). Least-Squares Estimation. From Gauss To Kalman. *IEEE Spectrum*, 7(7), 63–68.
- Syed, S., & Jansen, S. (2013). On Clusters in Open Source Ecosystems. *5th International Workshop on Software Ecosystems (IWSECO 2013)*, 13.
- Tao, Y. (2013). Ontology-based active repository system. *Information Technology Journal*, 12(11), 2138–2145.

- Tellis, G.J., & Crawford, C.M. (1981). An evolutionary approach to product growth theory. *The Journal of Marketing*, 125–132.
- Watts, D., & Strogatz, S. (1998). Collective dynamics of 'small-world' networks. *nature*, 393(6684), 440–442.
- Welch, B.L. (1947). The generalization of student's' problem when several different population variances are involved. *Biometrika*, 28–35.

Appendix A: Analysis Data

The sample set of ecosystems with a user as origin, with the corresponding network properties, is shown in the table below.

Table 9.1: Results for user-centered ecosystem networks.

#	Root/origin user	Regression polynomial: $f(x)$	Polynomial category	Average network density	Average degree centralization	Average clustering coefficient
1	coss	$-42.5x^4 + 90.5x^3 - 61.0x^2 + 13.3x + 0.1$	+-+--	0.08	0.30	0.46
2	kpieters	$-3.4x^4 + 3.6x^3 + 0.1x^2 - 1.0x + 0.7$	+----	0.11	0.29	0.47
3	cgmartin	$-24.8x^4 + 51.5x^3 - 36.8x^2 + 10.0x - 0.2$	+-+--	0.16	0.34	0.45
4	diemuzi	$-3.8x^4 + 14.0x^3 - 15.1x^2 + 4.7x + 0.5$	+----	0.11	0.39	0.53
5	WalterTamboer	$-4.3x^4 + 12.6x^3 - 13.0x^2 + 4.4x + 0.5$	+----	0.11	0.35	0.51
6	Slamdunk	$-18.3x^4 + 30.4x^3 - 14.5x^2 + 1.2x + 0.7$	--+--	0.11	0.31	0.37
7	yanickrochon	$-11.9x^4 + 19.4x^3 - 8.6x^2 + 1.2x + 0.5$	+++--	0.07	0.44	0.42
8	BilgeXA	$-22.6x^4 + 49.5x^3 - 33.6x^2 + 6.1x + 0.6$	--+--	0.08	0.10	0.23
9	raykolbe	$-7.8x^4 + 20.1x^3 - 18.3x^2 + 5.7x + 0.1$	----+	0.21	0.49	0.23
10	figof	$-15.0x^4 + 32.3x^3 - 22.3x^2 + 4.6x + 0.6$	+----	0.50	0.52	0.98
11	superdweebie	$-14.7x^4 + 30.9x^3 - 20.3x^2 + 3.7x + 0.2$	--+--	0.26	0.11	0.11
12	oppegard	$8.4x^4 - 22.8x^3 + 20.6x^2 - 6.3x + 0.6$	++-+	0.17	0.40	0.40
13	oparadis	$-9.4x^4 + 11.6x^3 - 0.2x^2 - 2.8x + 0.7$	++-+	0.09	0.29	0.17
14	Ph3nol	$-21.0x^4 + 43.1x^3 - 29.4x^2 + 7.5x + 0.2$	++---	0.05	0.35	0.41
15	michaelperrin	$-21.5x^4 + 43.2x^3 - 29.1x^2 + 7.7x + 0.2$	++---	0.05	0.27	0.39
16	gaurish	$8.9x^4 - 17.4x^3 + 7.8x^2 + 0.4x + 0.4$	++++	0.10	0.27	0.50
17	Lumbendil	$-25.3x^4 + 56.6x^3 - 40.3x^2 + 8.4x + 0.5$	--+--	0.22	0.11	0.17
18	michelsalib	$-40.8x^4 + 81.9x^3 - 51.3x^2 + 9.6x + 0.3$	--+--	0.14	0.19	0.18
19	cruelwen	$-8.1x^4 + 18.0x^3 - 11.6x^2 + 2.0x + 0.7$	++++	0.85	0.33	0.41
20	Hikkijp	$-2.7x^4 + 12.7x^3 - 15.2x^2 + 6.0x - 0.1$	++-+	0.35	0.32	0.61
21	bezhermoso	$-7.4x^4 + 16.0x^3 - 13.8x^2 + 5.0x + 0.3$	++++	0.05	0.31	0.39
22	erodataM	$1.8x^4 - 1.4x^3 - 2.4x^2 + 2.2x + 0.3$	++-+	0.08	0.51	0.34
23	syslwg	$-9.7x^4 + 21.6x^3 - 17.2x^2 + 6.3x - 0.2$	++-+	0.39	0.60	0.65
24	nomack84	$-25.2x^4 + 58.2x^3 - 41.9x^2 + 9.2x + 0.2$	--+--	0.03	0.16	0.24
25	davidwindell	$-40.5x^4 + 88.2x^3 - 61.6x^2 + 13.9x$	--+--	0.05	0.23	0.27
26	lmoehn	$-1.9x^4 + 10.6x^3 - 14.4x^2 + 6.0x + 0.1$	++++	0.41	0.41	0.62
27	khwang1	$-30.1x^4 + 58.6x^3 - 34.3x^2 + 6.6x - 0.2$	++-+	0.22	0.10	0.16
28	mattupstate	$3.4x^4 - 16.4x^3 + 16.8x^2 - 4.4x + 0.3$	++-+	0.66	0.23	0.39
29	ekonomiyaki3000	$-15.4x^4 + 35.2x^3 - 26.7x^2 + 7.0x - 0.2$	----	0.48	0.17	0.57
30	Baachi	$-29.5x^4 + 57.0x^3 - 32.0x^2 + 3.8x + 0.8$	--+--	0.06	0.18	0.20
31	aliismayilov	$-0.6x^4 - 1.1x^3 + 2.0x$	++-+	0.49	0.51	0.68
32	fukajun	$-3.2x^4 + 4.6x^3 - 2.7x^2 + 1.3x$	++-+	0.77	0.46	0.77

33	gedrox	$-12.4x^4 + 27.5x^3 - 20.4x^2 + 5.6x + 0.4$	++-+	0.06	0.29	0.41
34	hs Wong3i	$15.2x^4 - 31.3x^3 + 17.7x^2 - 1.1x$	++-+	0.22	0.48	0.41
35	Partugal	$-41.1x^4 + 84.8x^3 - 55.9x^2 + 11.8x + 0.1$	--++	0.06	0.18	0.24
36	marceloboeira	$-29.9x^4 + 55.4x^3 - 30.6x^2 + 5.4x - 0.2$	---+	0.46	0.18	0.20
37	pwenig	$-16.0x^4 + 40.6x^3 - 33.7x^2 + 9.1x + 0.2$	---+	0.29	0.51	0.38
38	twinturbo	$-34.2x^4 + 72.7x^3 - 50.0x^2 + 11.5x - 0.2$	---+	0.16	0.36	0.27
39	bar	$-37.4x^4 + 76.7x^3 - 49.8x^2 + 10.1x + 0.1$	---+	0.13	0.31	0.19
40	dhozac	$-7.7x^4 + 16.5x^3 - 13.6x^2 + 4.6x + 0.4$	----	0.55	0.53	0.81
41	julidendidier	$7.3x^4 - 17.7x^3 + 15.8x^2 - 6.2x + 0.9$	--++	0.42	0.14	0.10
42	aaronschmitz	$-24.7x^4 + 51.6x^3 - 33.5x^2 + 6.2x + 0.3$	---+	0.50	0.02	0.25
43	nclundsten	$-18.1x^4 + 38.9x^3 - 26.4x^2 + 5.2x + 0.2$	---+	0.30	0.16	0.17
44	franciscomxs	$32.1x^4 - 68.6x^3 + 45.3x^2 - 9.1x + 0.5$	-++-	0.28	0.19	0.21
45	kozo002	$-11.6x^4 + 24.6x^3 - 18.3x^2 + 4.7x + 0.5$	++-+	0.74	0.34	0.85
46	benlumley	$-29.9x^4 + 62.1x^3 - 41.9x^2 + 10.2x - 0.2$	++++	0.08	0.31	0.36
47	TomAdam	$-20.1x^4 + 48.5x^3 - 35.1x^2 + 7.2x + 0.3$	---+	0.07	0.28	0.23
48	andreatarr	$-17.2x^4 + 37.3x^3 - 25.2x^2 + 4.6x + 0.6$	---+	0.46	0.59	0.64
49	vbardales	$-12.3x^4 + 26.8x^3 - 19.6x^2 + 5.1x + 0.2$	++-+	0.07	0.35	0.36
50	pasamio	$-24.1x^4 + 44.4x^3 - 24.3x^2 + 4.0x$	---+	0.34	0.17	0.38
51	glowell2	$-4.1x^4 + 11.0x^3 - 10.5x^2 + 3.8x + 0.3$	++-+	0.57	0.49	0.72
52	shageman	$-0.4x^4 + 1.9x^3 - 3.8x^2 + 2.3x + 0.5$	++-+	0.29	0.75	0.75
53	oszczep	$4.2x^4 - 4.0x^3 - 4.8x^2 + 4.7x$	++-+	0.42	0.66	0.53
54	johanpoirier	$-18.6x^4 + 35.5x^3 - 21.7x^2 + 5.2x + 0.2$	++-+	0.78	0.29	0.80
55	vandosant	$-25.4x^4 + 57.2x^3 - 43.6x^2 + 12.1x - 0.1$	---+	0.35	0.55	0.53
56	andras kende	$-25.7x^4 + 46.3x^3 - 23.6x^2 + 2.6x + 0.6$	---+	0.29	0.33	0.51
57	jtuchscherer	$-7.2x^4 + 13.8x^3 - 7.7x^2 + 1.2x + 0.4$	++-+	0.25	0.48	0.61
58	mikeryan-inktank	$2.7x^4 + 0.9x^3 - 3.8x^2 + 0.8x + 0.5$	++++	0.40	0.30	0.43
59	tlabeeuw	$18.3x^4 - 36.0x^3 + 20.6x^2 - 2.6x + 0.3$	++-+	0.26	0.57	0.58
60	seishingithub	$-6.3x^4 + 18.8x^3 - 19.6x^2 + 7.4x + 0.1$	++-+	0.29	0.42	0.60
61	theodorDiaconu	$-5.8x^4 + 17.6x^3 - 17.1x^2 + 5.2x + 0.4$	++-+	0.72	0.34	0.66
62	acasademont	$-29.6x^4 + 62.7x^3 - 41.5x^2 + 7.9x + 0.6$	---+	0.12	0.14	0.29
63	jumph4x	$4.5x^4 - 10.7x^3 + 9.3x^2 - 3.6x + 0.6$	--++	0.17	0.15	0.09
64	krystalcampioni	$-2.0x^4 - 2.3x^3 + 7.8x^2 - 3.9x + 0.6$	---+	0.30	0.24	0.38
65	andreibondarev	$6.6x^4 - 3.4x^3 - 5.4x^2 + 2.5x + 0.5$	++++	0.31	0.37	0.39
66	danielholmes	$18.2x^4 - 38.4x^3 + 26.8x^2 - 7.3x + 0.8$	--++	0.12	0.08	0.09
67	NARKOZ	$-28.6x^4 + 60.7x^3 - 40.9x^2 + 8.2x + 0.3$	---+	0.33	0.35	0.24
68	ir3	$-0.6x^4 - 0.3x^3 + 0.9x^2 + 0.2$	-++-	0.75	0.41	0.87
69	fazy	$-36.2x^4 + 68.3x^3 - 38.2x^2 + 6.4x + 0.1$	---+	0.57	0.22	0.58
70	AndreasMaier	$-16.6x^4 + 31.9x^3 - 20.5x^2 + 5.3x + 0.4$	++-+	0.26	0.73	0.78
71	matteocaberlotto	$0.6x^4 - 3.5x^3 + 5.1x^2 - 2.7x + 0.5$	--++	0.49	0.04	0.04
72	stevepm	$6.3x^4 - 7.8x^3 - 1.7x^2 + 3.3x + 0.3$	++-+	0.37	0.35	0.56
73	iwhurtafly	$-0.6x^4 - 0.3x^3 + 0.9x^2 + 0.2$	-++-	0.85	0.39	0.95
74	siong1987	$-18.9x^4 + 34.7x^3 - 20.5x^2 + 4.9x + 0.2$	++-+	0.07	0.24	0.32
75	Kimundi	$-4.2x^4 + 1.8x^3 + 4.4x^2 - 1.9x + 0.4$	++-+	0.09	0.28	0.56
76	dongilbert	$9.7x^4 - 16.6x^3 + 9.0x^2 - 1.6x + 0.1$	++++	0.38	0.34	0.20
77	dengwa	$-30.8x^4 + 55.4x^3 - 29.1x^2 + 5.0x - 0.2$	---+	0.51	0.39	0.46
78	durhamka	$24.9x^4 - 51.1x^3 + 32.3x^2 - 6.2x + 0.3$	-++-	0.37	0.27	0.29
79	acapilleri	$-16.4x^4 + 32.8x^3 - 22.8x^2 + 6.3x + 0.4$	++-+	0.08	0.25	0.51
80	rufinus	$-18.8x^4 + 39.5x^3 - 29.5x^2 + 8.8x + 0.1$	---+	0.07	0.27	0.46
81	keqh	$-16.7x^4 + 32.8x^3 - 20.0x^2 + 4.2x + 0.3$	++-+	0.85	0.39	0.95
82	thanosp	$-29.9x^4 + 63.4x^3 - 42.8x^2 + 9.7x - 0.2$	++++	.12	0.26	0.22
83	pivotal-vmware	$-23.0x^4 + 47.1x^3 - 30.0x^2 + 6.3x + 0.2$	++-+	0.39	0.62	0.70
84	xtian	$-0.7x^4 - 3.5x^3 + 7.4x^2 - 4.3x + 1.0$	--++	0.10	0.31	0.26
85	alopropoz	$-7.6x^4 + 18.1x^3 - 15.4x^2 + 4.4x + 0.5$	++-+	0.61	0.53	0.73
86	ezkl	$5.5x^4 - 14.2x^3 + 13.2x^2 - 5.1x + 0.7$	--++	0.30	0.26	0.03
87	blois	$-8.8x^4 + 17.4x^3 - 14.6x^2 + 5.7x + 0.2$	----	0.44	0.36	0.49
88	XenoPhex	$3.4x^4 - 7.7x^3 + 8.4x^2 - 3.6x + 0.5$	---+	0.40	0.59	0.19
89	spencereldred	$-31.7x^4 + 64.7x^3 - 41.5x^2 + 8.0x + 0.2$	---+	0.51	0.50	0.28
90	bsodmike	$-23.2x^4 + 44.7x^3 - 28.8x^2 + 7.1x + 0.1$	++-+	0.09	0.24	0.34
91	chou	$-30.0x^4 + 65.3x^3 - 45.1x^2 + 9.5x + 0.2$	---+	0.36	0.18	0.36
92	benjaminpick	$-32.8x^4 + 66.0x^3 - 40.6x^2 + 8.2x - 0.2$	---+	0.60	0.08	0.42
93	makaroni4	$3.4x^4 - 2.2x^3 - 0.4x^2 + 0.3x$	++++	0.25	0.12	0.07
94	tdboone	$-46.6x^4 + 90.8x^3 - 53.9x^2 + 9.9x$	---+	0.38	0.25	0.38
95	jcpivotallabs	$-16.3x^4 + 30.8x^3 - 19.4x^2 + 4.8x + 0.4$	++-+	0.30	0.36	0.68
96	wandtasie	$-6.3x^4 + 14.5x^3 - 11.1x^2 + 2.7x + 0.4$	++-+	0.13	0.36	0.26
97	apperly	$-10.9x^4 + 18.9x^3 - 10.6x^2 + 3.0x + 0.1$	++-+	0.31	0.56	0.64

98	ayrton	$-16.0x^4 + 31.6x^3 - 21.6x^2 + 5.9x + 0.4$	++--	0.08	0.24	0.50
99	Dinduks	$-22.6x^4 + 43.4x^3 - 26.5x^2 + 5.9x + 0.3$	++--	0.04	0.32	0.41
100	dliebreich	$-4.1x^4 + 10.5x^3 - 7.4x^2 + 1.7x + 0.3$	----	0.23	0.32	0.53
101	cf-bosh	$-6.9x^4 + 10.6x^3 - 4.4x^2 + 0.8x + 0.3$	++--	0.35	0.61	0.67
102	jcorcuera	$-11.7x^4 + 23.3x^3 - 16.2x^2 + 4.8x + 0.3$	++--	0.07	0.24	0.43
103	ems2141	$10.9x^4 - 9.6x^3 - 5.9x^2 + 4.9x + 0.3$	++--	0.25	0.26	0.49
104	kushmerick	$2.9x^4 - 4.4x^3 - 0.7x^2 + 2.3x + 0.3$	++--	0.40	0.53	0.77
105	JennyAllar	$-18.5x^4 + 48.8x^3 - 41.8x^2 + 12.0x - 0.1$	++--	0.38	0.26	0.57
106	mikekauffman	$-6.6x^4 + 26.1x^3 - 28.2x^2 + 9.2x + 0.1$	++--	0.25	0.32	0.49
107	Romain-Geissler	$-12.8x^4 + 31.9x^3 - 24.9x^2 + 6.1x + 0.3$	++--	0.05	0.23	0.32
108	brannon	$2.4x^4 - 1.9x^3 + 0.2x + 0.1$	----	0.79	0.47	0.64
109	route	$-44.1x^4 + 86.4x^3 - 51.3x^2 + 8.9x + 0.2$	----	0.17	0.24	0.25
110	lunks	$-5.4x^4 + 10.6x^3 - 7.1x^2 + 2.0x - 0.1$	++--	0.46	0.32	0.33
111	wzzrd	$28.0x^4 - 49.3x^3 + 22.6x^2 - 0.8x - 0.1$	----	0.60	0.28	0.32
112	cf-runtime	$-10.9x^4 + 21.8x^3 - 14.6x^2 + 3.9x + 0.5$	++--	0.29	0.44	0.71
113	guilsa	$-27.1x^4 + 56.0x^3 - 36.7x^2 + 7.8x + 0.1$	++--	0.55	0.38	0.65
114	giosh94mhz	$-1.0x^4 + 6.0x^3 - 9.8x^2 + 5.0x + 0.1$	++--	0.04	0.29	0.30
	Average	$-11.7x^4 + 24.6x^3 - 16.9x^2 + 4.0x + 0.3$		0.30	0.33	0.44

The sample set of ecosystems with a repository as origin, with the corresponding network properties, is shown in the table below.

Table 9.2: Results for repository-centered ecosystem networks.

#	Root/origin repository	Regression polynomial: $f(x) =$	Polynomial category	Average network density	Average degree centralization	Average clustering coefficient
1	yapplabs/glazier	$12.7x^4 - 26.3x^3 + 16.1x^2 - 2.4x + 0.3$	++--	0.40	0.47	0.76
2	tooling/authoring-styleguide	$-16.0x^4 + 30.1x^3 - 18.6x^2 + 4.9x + 0.1$	++--	0.72	0.20	0.85
3	angular/angular-sites	$-5.5x^4 + 11.5x^3 - 10.6x^2 + 5.0x$	++--	0.56	0.55	0.77
4	kevva/imagemin-gui	$-17.1x^4 + 36.2x^3 - 25.9x^2 + 6.5x + 0.4$	++--	0.22	0.47	0.74
5	clear-code/internship	$-22.8x^4 + 54.4x^3 - 43.0x^2 + 11.6x - 0.1$	----	0.67	0.56	0.53
6	joliss/broccoli	$-3.4x^4 + 9.7x^3 - 11.8x^2 + 5.2x + 0.3$	++--	0.22	0.33	0.79
7	dart-lang/dartlang.org	$4.9x^4 - 11.2x^3 + 9.1x^2 - 2.9x + 0.4$	++--	0.60	0.37	0.23
8	c9/runjs	$-14.4x^4 + 33.4x^3 - 24.4x^2 + 5.4x + 0.2$	++--	0.12	0.34	0.27
9	saltstack-formulas/tomcat-formula	$-15.8x^4 + 30.5x^3 - 20.5x^2 + 5.3x + 0.4$	----	0.47	0.52	0.86
10	mozilla/gaia-ui-tests	$20.5x^4 - 38.8x^3 + 23.9x^2 - 5.4x + 0.8$	++++	0.45	0.49	0.63
11	droonga/express-droonga	$-0.9x^4 - 3.5x^3 + 7.2x^2 - 2.6x + 0.5$	++--	0.47	0.31	0.69
12	snowplow/referer-parser	$-13.8x^4 + 26.5x^3 - 17.1x^2 + 4.6x + 0.3$	++--	0.82	0.20	0.84
13	tinkerpop/furnace	$-25.2x^4 + 55.4x^3 - 38.9x^2 + 8.7x + 0.1$	++--	0.47	0.21	0.43
14	thinkarelius/faunus	$6.5x^4 - 4.0x^3 - 4.7x^2 + 2.9x + 0.1$	++--	0.39	0.16	0.31
15	sindresorhus/get-std	$-16.1x^4 + 30.1x^3 - 18.9x^2 + 5.5x - 0.1$	++--	0.13	0.29	0.72
16	dockyard/ember-appkit-rails	$1.0x^4 + 0.2x^3 - 0.6x^2 - 0.1x + 0.3$	++++	0.29	0.31	0.54
17	ContinuumIO/conda	$24.0x^4 - 48.7x^3 + 31.9x^2 - 7.1x + 0.5$	++--	0.57	0.28	0.08
18	koajs/send	$-19.2x^4 + 40.4x^3 - 28.0x^2 + 6.8x + 0.4$	++--	0.83	0.13	0.88
19	atom/open-on-github	$7.3x^4 - 11.6x^3 + 8.3x^2 - 3.4x + 0.6$	++++	0.18	0.13	0.27
20	WebComponentsOrg/webcomponents.org	$-17.2x^4 + 40.2x^3 - 29.7x^2 + 6.2x + 0.5$	++--	0.38	0.23	0.31
21	isaacs/read	$27.0x^4 - 56.7x^3 + 37.4x^2 - 8.5x + 0.9$	----	0.35	0.21	0.32
22	addyosmani/traceur-todomvc	$-14.9x^4 + 28.6x^3 - 18.9x^2 + 5.0x + 0.4$	----	0.63	0.42	0.93
23	sindresorhus/slang-haven	$-27.2x^4 + 54.1x^3 - 31.9x^2 + 4.7x + 0.7$	++--	0.43	0.48	0.71
24	ebryn/bugzilla-ember	$-12.7x^4 + 25.4x^3 - 16.6x^2 + 4.3x + 0.2$	++--	0.45	0.52	0.85
25	groonga/gcs	$-0.8x^4 - 2.1x^3 + 5.7x^2 - 3.2x + 0.6$	++--	0.54	0.40	0.70
26	stefanpenner/ember-app-kit	$-11.1x^4 + 22.5x^3 - 12.8x^2 + 1.5x + 0.4$	++++	0.29	0.41	0.53
27	ajaxorg/node-github	$-2.2x^4 + 6.8x^3 - 8.0x^2 + 3.3x + 0.1$	----	0.32	0.53	0.50

28	CocoaPods/beta.cocoapods.org	$-10.8x^4 + 28.0x^3 - 25.4x^2 + 8.7x - 0.1$	+-+ +	0.56	0.45	0.76
29	ipython/nbconvert	$-1.9x^4 + 4.0x^3 - 2.2x^2 - 0.3x + 0.5$	--+ +	0.39	0.34	0.25
30	react-php/react	$-17.7x^4 + 30.8x^3 - 14.9x^2 + 2.3x - 0.1$	-+ + -	0.54	0.14	0.61
31	yahoo/gear	$-31.5x^4 + 58.4x^3 - 32.7x^2 + 6.0x - 0.2$	-+ + -	0.41	0.14	0.17
32	sameera2004/NLSLS2	$-17.1x^4 + 27.6x^3 - 12.8x^2 + 2.5x$	+ + - -	0.50	0.31	0.71
33	LearnBoost/monk	$-5.4x^4 + 14.4x^3 - 14.5x^2 + 5.8x - 0.1$	-+ + +	0.14	0.36	0.30
34	IndigoUnited/automaton	$-5.4x^4 + 10.5x^3 - 6.6x^2 + 1.6x$	-+ + +	0.62	0.14	0.44
35	driftyco/ionic-angular-cordova-seed	$11.9x^4 - 26.7x^3 + 19.1x^2 - 4.0x + 0.4$	+ + - +	0.71	0.35	0.79
36	component/droplod	$9.2x^4 - 15.0x^3 + 3.6x^2 + 1.6x + 0.6$	+ + - +	0.58	0.33	0.70
37	githubtrainer/poetry	$15.4x^4 - 26.0x^3 + 12.4x^2 - 1.7x + 0.4$	-+ + +	0.51	0.37	0.20
38	aurajs/aura	$12.0x^4 - 22.1x^3 + 11.7x^2 - 1.7x + 0.2$	-+ + +	0.48	0.24	0.38
39	logaling/logaling-server	$-5.3x^4 + 11.9x^3 - 8.8x^2 + 2.1x + 0.1$	-+ + +	0.83	0.27	0.34
40	bigpipe/pipe.js	$-6.3x^4 + 5.8x^3 + 1.1x^2 - 0.8x + 0.4$	+ + - +	0.83	0.16	0.82
41	github/teach.github.com	$0.1x^4 + 1.7x^3 - 3.4x^2 + 1.6x + 0.2$	-+ + +	0.83	0.25	0.30
42	Sylius/SyliusAssortmentBundle	$8.0x^4 - 19.1x^3 + 15.5x^2 - 4.9x + 0.5$	-+ + +	1.09	0.05	1.12
43	alchemy-fr/PHP-dataURI	$-22.9x^4 + 51.2x^3 - 36.3x^2 + 8.9x - 0.3$	-+ + -	0.44	0.30	0.48
44	3rd-Eden/bigpipe	$4.6x^4 - 6.3x^3 + 3.9x^2 - 1.9x + 0.6$	-+ + +	0.72	0.13	0.47
45	willdurand/BazingaExposeTranslationBundle	$5.4x^4 - 11.8x^3 + 7.5x^2 - 0.9x + 0.6$	+ + - +	0.03	0.39	0.40
46	pandamicro/plugin-x	$-30.4x^4 + 59.0x^3 - 35.3x^2 + 6.7x + 0.5$	+ + + -	0.33	0.56	0.49
47	yapplabs/mhe-metadata-js	$-20.6x^4 + 38.7x^3 - 23.7x^2 + 5.8x + 0.2$	+ + - -	0.41	0.41	0.89
48	janestreet/patience_diff	$-12.0x^4 + 24.6x^3 - 13.8x^2 + 2.1x$	-+ + -	0.51	0.11	0.55
49	johnmccutchan/markerprof	$-4.2x^4 + 9.8x^3 - 8.5x^2 + 3.1x + 0.1$	-+ + +	1.19	0.15	1.19
50	zwaldowski/AZCoreRecord	$5.5x^4 - 11.2x^3 + 8.8x^2 - 3.2x + 0.6$	-+ + +	0.30	0.25	0.22
51	CakeDC/markup_parsers	$-3.0x^4 + 12.3x^3 - 14.0x^2 + 4.3x + 0.6$	+ + + +	0.30	0.47	0.51
52	filamentgroup/tappy	$-19.9x^4 + 38.8x^3 - 22.3x^2 + 2.4x + 0.7$	-+ + -	0.57	0.10	0.46
53	radio-tools/spectral-cube	$9.4x^4 - 20.7x^3 + 10.9x^2 + 0.2x + 0.2$	+ + - +	0.35	0.52	0.48
54	hull/minimihull	$-7.3x^4 + 19.7x^3 - 19.1x^2 + 6.9x + 0.2$	+ + - +	0.70	0.50	0.90
55	jspahrsummers/xconfgs	$15.5x^4 - 29.5x^3 + 18.2x^2 - 4.0x + 0.5$	-+ + +	0.27	0.14	0.32
56	phinze/homebrew-cask	$-19.6x^4 + 37.2x^3 - 21.5x^2 + 4.1x - 0.2$	-+ + -	0.72	0.37	0.32
57	digitalbazaar/payswarm.js	$12.7x^4 - 29.7x^3 + 24.0x^2 - 7.3x + 0.8$	-+ + +	0.61	0.14	0.44
58	primus/metroplex	$-28.8x^4 + 58.3x^3 - 36.9x^2 + 7.7x - 0.2$	-+ + -	0.70	0.11	0.68
59	opsmezzo/composer-systems	$-29.4x^4 + 62.4x^3 - 39.5x^2 + 6.9x + 0.4$	-+ + -	0.36	0.25	0.44
60	orta/ARAnalytics	$3.3x^4 - 8.6x^3 + 9.3x^2 - 4.4x + 0.8$	-+ + +	0.22	0.28	0.32
61	MantleFramework/Mantle	$-17.9x^4 + 29.9x^3 - 15.8x^2 + 3.8x - 0.1$	-+ - -	0.59	0.27	0.48
62	NLSLS-II/userpackages	$-12.1x^4 + 15.6x^3 - 4.0x^2 + 0.5x + 0.1$	-+ - -	0.67	0.24	0.81
63	vojtajina/testacular	$8.2x^4 - 16.7x^3 + 10.6x^2 - 2.2x + 0.1$	-+ - +	0.37	0.28	0.17
64	Gitonomy/gitonomy	$-2.9x^4 + 2.2x^3 + 3.4x^2 - 3.6x + 0.8$	-+ + +	0.44	0.17	0.13
65	Vluxe/Orca	$-13.2x^4 + 27.7x^3 - 16.9x^2 + 3.3x - 0.1$	-+ + -	0.97	0.21	1.04
66	adrm/astropy	$-15.4x^4 + 29.4x^3 - 18.8x^2 + 4.4x + 0.5$	-+ - -	0.54	0.58	0.84
67	12sm/depuy	$-26.2x^4 + 48.2x^3 - 27.8x^2 + 6.1x + 0.1$	+ + - -	0.87	0.30	0.82
68	f/awesome-safran	$-24.2x^4 + 49.5x^3 - 33.8x^2 + 8.4x + 0.4$	+ - - +	0.72	0.49	0.77
69	jzaefferer/grunt-jquery-content	$-9.4x^4 + 15.0x^3 - 8.7x^2 + 2.2x + 0.6$	+ + - +	0.31	0.49	0.68
70	Numbee/Optimal	$-20.3x^4 + 39.4x^3 - 25.9x^2 + 7.1x + 0.1$	+ + - -	1.24	0.00	1.24
71	bocoup/gaia	$-0.6x^4 - 4.6x^3 + 6.5x^2 - 1.3x + 0.1$	-+ - +	0.30	0.92	0.07
72	bobandraa/Coconnect-Four	$-14.8x^4 + 30.4x^3 - 22.8x^2 + 7.5x$	+ + + -	0.37	0.66	0.00
73	ding2tal/latto	$-11.7x^4 + 32.8x^3 - 29.3x^2 + 9.2x - 0.2$	+ + + +	0.61	0.47	0.64
74	loopdk/loop_frontend	$-15.9x^4 + 29.1x^3 - 16.4x^2 + 3.1x + 0.4$	+ + - -	0.53	0.45	0.73
75	aakb/odaa_drupal	$-22.7x^4 + 46.3x^3 - 30.9x^2 + 7.2x + 0.2$	+ + - +	0.64	0.35	0.80
76	telefonicaid/wakeup_platform_common	$-3.5x^4 + 11.7x^3 - 9.6x^2 + 2.4x - 0.1$	-+ + +	0.91	0.18	0.79
77	uddannelse-laering-forloeb/ulftheme	$-24.5x^4 + 46.3x^3 - 27.7x^2 + 5.7x + 0.2$	+ + - -	0.63	0.35	0.72
78	emberui/emberui	$4.2x^4 - 4.1x^3 - 1.9x^2 + 2.1x - 0.1$	-+ + +	0.93	0.12	0.29
79	pascalchevrel/webdashboard	$-11.9x^4 + 27.8x^3 - 19.5x^2 + 4.1x + 0.4$	+ + + -	0.30	0.46	0.46
80	scottgonzalez/pretty-diff	$-39.0x^4 + 80.2x^3 - 50.9x^2 + 10.5x - 0.1$	-+ + -	0.24	0.31	0.35
81	jquery/contribute.jquery.com	$-23.7x^4 + 43.4x^3 - 24.1x^2 + 4.5x + 0.4$	+ + + -	0.30	0.47	0.55
82	12-oz/pliny	$-7.2x^4 + 12.8x^3 - 6.3x^2 + 1.4x + 0.1$	-+ + +	0.76	0.22	0.49
83	vigo/ruby101-kitap	$-11.7x^4 + 25.4x^3 - 19.4x^2 + 5.7x + 0.4$	+ + - +	0.64	0.49	0.72
84	cascadiajs/cascadiajs.github.com	$8.8x^4 - 18.1x^3 + 8.6x^2 + 0.5x + 0.2$	-+ - +	0.35	0.44	0.10
85	ddollar/heroku-buildpacks	$-20.1x^4 + 43.5x^3 - 30.9x^2 + 8.0x + 0.2$	+ + + -	0.68	0.63	0.51
86	overtone/overtone	$-14.7x^4 + 29.5x^3 - 21.1x^2 + 6.3x + 0.1$	+ + + -	0.71	0.71	0.00
87	mcav/gaia-email-libs-and-more	$-16.3x^4 + 33.7x^3 - 24.6x^2 + 7.0x + 0.3$	+ + - +	0.07	0.56	0.56
88	emberjs/docs-generator	$-48.4x^4 + 98.6x^3 - 63.4x^2 + 12.9x + 0.2$	+ + - -	0.14	0.73	0.31
89	cgjones/platform-demo-mc	$-8.7x^4 + 22.9x^3 - 19.8x^2 + 5.7x + 0.4$	-+ - +	0.15	0.60	0.36
90	albertopq/gaia	$-11.2x^4 + 21.3x^3 - 12.2x^2 + 2.0x + 0.5$	+ + + +	0.17	0.82	0.30
91	heroku/heroku-buildpacks	$-41.7x^4 + 79.4x^3 - 45.5x^2 + 7.7x + 0.2$	-+ + -	0.46	0.27	0.42
92	mozilla-b2g/android-device-unagi	$-4.3x^4 + 12.1x^3 - 11.1x^2 + 2.6x + 0.7$	+ + - +	0.18	0.58	0.33

93	feincms/feincms	$20.8x^4 - 40.7x^3 + 24.1x^2 - 4.2x + 0.2$	----+	0.58	0.20	0.17
94	heroku/buildkits	$-27.8x^4 + 55.3x^3 - 33.1x^2 + 5.4x + 0.6$	+--	0.66	0.50	0.47
95	alex/cryptography	$-7.1x^4 + 4.5x^3 + 5.6x^2 - 2.9x + 0.4$	+++	0.82	0.18	0.62
96	freshbooks/ember-responsive	$8.5x^4 - 13.8x^3 + 1.4x^2 + 4.0x - 0.1$	++++	0.34	0.27	0.37
97	rnowm/Gaia-UI-Building-Blocks	$-18.4x^4 + 33.4x^3 - 19.5x^2 + 4.0x + 0.4$	----	0.30	0.49	0.36
98	terminalmage/django-tutorial	$-13.8x^4 + 28.5x^3 - 18.7x^2 + 3.5x + 0.6$	+++	0.50	0.78	0.82
99	influxdb/influxdb-ruby	$-4.8x^4 + 9.5x^3 - 5.2x^2 + 0.6x + 0.2$	+++	0.71	0.18	0.71
100	asutherland/bleach.js	$-3.1x^4 + 16.0x^3 - 17.5x^2 + 5.2x + 0.2$	++++	0.23	0.42	0.38
101	mozsqib/jsas	$-10.2x^4 + 18.8x^3 - 12.4x^2 + 3.4x + 0.5$	++++	0.16	0.97	0.09
102	cloudkeep/symantecssl	$-11.9x^4 + 18.9x^3 - 6.0x^2 - 1.8x + 0.9$	----	0.42	0.39	0.16
103	dualface/cocos2d-x	$-51.1x^4 + 110.1x^3 - 76.9x^2 + 18.2x - 0.3$	----	0.67	0.23	0.45
104	cdnjs/autoupdate	$-33.3x^4 + 65.3x^3 - 38.7x^2 + 6.9x + 0.2$	----	0.53	0.57	0.48
105	rpfflorence/ember-qunit	$-26.6x^4 + 65.2x^3 - 51.1x^2 + 13.1x - 0.1$	----	0.10	0.26	0.43
106	jsonresume/resumeToMarkdown	$-30.0x^4 + 64.4x^3 - 43.9x^2 + 9.4x + 0.1$	----	0.54	0.30	0.41
107	dominictarr/through	$-21.8x^4 + 51.2x^3 - 39.1x^2 + 9.5x + 0.2$	----	0.48	0.24	0.55
108	savoirfairelinux/plugin-check_printer_hp_2600n	$-22.3x^4 + 41.9x^3 - 25.8x^2 + 6.4x + 0.1$	----	0.68	0.23	0.90
109	cloudkeep-ops/barbican-postgresql	$-23.8x^4 + 51.5x^3 - 37.6x^2 + 10.1x - 0.1$	++-	0.81	0.22	0.76
110	yawnt/bees	$-29.0x^4 + 57.6x^3 - 35.7x^2 + 6.6x + 0.6$	+-	0.68	0.35	0.55
111	meshy/pythonwheels	$-11.4x^4 + 20.0x^3 - 8.9x^2 - 0.2x + 0.8$	+-	0.79	0.21	0.78
112	saltstack/salt-cloud	$-13.2x^4 + 24.7x^3 - 14.8x^2 + 3.3x$	----	0.57	0.59	0.49
113	urlship/Owl	$-16.2x^4 + 33.0x^3 - 20.2x^2 + 2.5x + 0.8$	----	1.06	0.00	0.51
114	ProjectMeniscus/meniscus	$-11.7x^4 + 22.0x^3 - 12.9x^2 + 2.9x + 0.2$	+++	0.85	0.27	0.53
	Average:	$-10.3x^4 + 21.0x^3 - 13.9x^2 + 3.2x + 0.3$		0.52	0.35	0.53