Steering Insight: An exploration of the Ruby Software Ecosystem

Jaap Kabbedijk and Slinger Jansen

Utrecht University
Department of Information and Computing Sciences
Princetonplein 5, 3584CC, Utrecht, Netherlands
{j.kabbedijk,s.jansen}@cs.uu.nl

Abstract. Software products are part of a larger network of products, suppliers and partners, called a *software ecosystem*, working together in order to provide functionality for the users and generate profit for the vendors. Not much is known about the characteristics and relationships within such a software ecosystem. This paper presents an overview of the open source Ruby ecosystem and lists its elements, characteristics, descriptives, roles, cliques and relationships. Data is gathered using the Git decentralized source code management system and is analyzed using social network and statistical analysis techniques. Our analysis shows that the Ruby ecosystem exists out of a couple very distinctive roles developers fulfil. It also shows that within the Ruby ecosystem only a small 'core' of approximately 10% of all developers and gems (Ruby packages) are dominant within the ecosystem. At this point in time it appears that the rails community would benefit from motivating current developers to work together more, instead of supporting new developers or gems in order to get a healthy ecosystem.

Key words: software ecosystem, Ruby, ecosystem governance, exploratory case study

1 Introduction

As a software vendor, your profit is not determined by one independent product, but by all parties and products related to your product. The development of software has a lot of differences compared to how physical products (e.g. bread or furniture) are created. First of all software is not one physical product, but a product that can be multiplied an infinite amount of time without substantial extra costs [1]. Second and most importantly, the total value of a software product is determined by sum of all additional products using the software product. The Android mobile phone operating system for example, has a limited value as a product on its own. The overall value of Android is determined by all the applications built on top of Android that extend the possibilities of the main software product (software platform). This network of all products, companies and services working together in one big network is called a Software Ecosystem. The term Software Ecosystem (SECO) was first coined by Messerschmitt

and Szyperski in 2005 [2], but it took until 2009 before a clear definition was formed by Jansen, Finkelstein and Brinkkemper [3], who define a SECO as "a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them".

"Making profit from your software product" became "making profit from your software ecosystem" [4]. Profit is not generated only by one product or service, but by all companies together making use of one core service to deliver their own products. The popularity of a product created by one company can benefit all companies and products in the same SECO. Software companies are increasingly becoming aware of this development and start to recognize their position within the SECO. Software vendors face new problems, as they need insight in their software ecosystems. They need to know the dynamics within their SECO, the participants and how to steer their SECO in order to generate the most profit. Jansen et al. [3] state that to the extent of getting a good insight into a SECO, characteristics must be quantified and measured. Possible characteristics are the number of sub communities, the reciprocity of the ecosystem or the outdegree of keystone actors. Iansiti and Levien [5] state that: "the performance of a firm is a function not only of its own capabilities or of its static position with respect to its competitors, customer, partners and suppliers, but of its dynamic interactions with the ecosystem as a whole". This statement emphasizes the importance of getting an overview of the entire SECO in order to judge the position of one participant in it.

Open source projects are typical environments in which SECOs develop around the community. By open source we mean that the source code of the project is available for download and everyone can participate in extending or adapting the code [6]. Among Free Open Source Software (FOSS) projects, relationships are frequently seen in the form of developers coding on multiple projects [7]. These developers are linchpins [8] between the different projects and are an important reason SECOs exists within FOSS. Since everyone can participate in coding for a specific project, large projects and with that, large interconnected SECOs come into play. Because of the size and level of transparency of these FOSS SECOs, they are ideal for statistical and network analysis, since a high sample size increases the significance of found results and improves the external validity [9]. Also, the use of large central repositories for storing all related projects enables a convenient way for gathering all data needed [10].

In this paper we first explain the main research question, including all sub questions related to this question in section 2. In (section 3) the Ruby case that was analyzed is described and the data gathering is explained. The research questions asked are analyzed into depth in section 4 and the results are presented in section 5. We conclude with the conclusion and discussion, followed by future research in section 6.

2 Research Questions

The main research questions answered in this paper is: **"What are the defining characteristics of a large scale open source software ecosystem?"**.
From the main research question, the following sub questions are derived:

1. **What elements can be identified within a SECO?** - An ecosystem exists out of several different elements interacting with each other. We expect different *types* of elements to exist within a SECO, each having different *functions* within the SECO. This question will be answered by looking at the description of all elements in the repository.

2. **What are the characteristics of the identified elements?** - Each of the defining characteristics for the SECO are required for the analysis of an element. A characteristics is defined as a prominent attribute of a SECO element.

3. **What are the descriptives of a SECO?** - In order to get a good overview of the SECO we are analyzing, we first need to explore the elements and characteristics within a SECO into some more depth. We focus on the software supply network level [3] when answering this question. Descriptives describe the main characteristics of a collection of the SECO elements quantitatively [11]

4. **What roles can be identified within a SECO?** - The elements existing within a SECO have different roles depending on their position in the SECO. We will identify several distinctive roles within a SECO based on their interconnectivity.

These sub questions will be answered in the following sections, concluded with the answer to the main research question in section 5.

3 Case Description and Data Gathering

In this research the relationships, dynamics and characteristics of the Ruby Ecosystem are analyzed. Ruby is a popular programming language, using the MVC design principle [12] and trying to combine a scripting language's simple immediacy (e.g. PHP) with a strict object oriented architecture (e.g. Java) [13]. The framework consists out of thousands of possible parts that can be combined to get exactly the functionality your project needs. The 'parts' are called 'gems' in Ruby jargon and are hosted and maintained through Git. Git is a Decentralized Source Code Management (DSCM) system [14] that uses decentralized repositories, owned by developers, that can be mined for analysis in the case of Ruby. Because all gems related the the Ruby programming language are released under an open source license [15], the entire Ruby ecosystem is a collection of FOSS projects. Gems are developed by different developers and everyone can

create as many gems as he wants. All gems are indexed on a central place and users of the Ruby programming language can decide which gems they need for their project development.

All data for this project was gathered by using an XML [16] overview of all gems present in the Ruby Git repository on *15-02-2010*. This XML contained information on each gem such as the name, dependencies, author, etc. Please see figure 1 for an excerpt of the XML file showing the XML data for one gem.

```
<rubygem>
  <dependencies>
    <runtime type="array"/>
    <development type="array"/>
  </dependencies>
  <name>activesupport</name>
  <downloads type="integer">377251</downloads>
  <info>Utility library which carries commonly used classes and goodies from the Rails framework</info>
  <version-downloads type="integer">253313</version-downloads>
  <version>2.3.5</version>
  <gem-uri>http://gemcutter.org/gems/activesupport-2.3.5.gem</gem-uri>
  <project-uri>http://gemcutter.org/gems/activesupport</project-uri>
  <authors>David Heinemeier Hansson</authors>
</rubygem>
```

Fig. 1: XML excerpt

By using the URI for the gem, the gem was downloaded from RubyGems[1] and stored in a a relational database. All additional meta information on the gem like the version of the gem release and developers who worked on the gem were stored in the database as well. This data gathering was done by using XSLT to get the right URI and the PHP scripting language to download the gems. The gems were downloaded to be able to analyze the source code of the gems and not only the meta information available of the gems. All data was stored in the database by using PHP scripting combined with SQL statements.

Before the data was usable for analysis, it had to be checked and reformatted. Due to errors in the XML source file and small glitches during data gathering, some data are incomplete or duplicated. We used the PASW analytics suite to identify errors and irregularities in the dataset[2] and corrected the gathered data based on this. The alterations to the original data are documented in our case study protocol [9].

4 Analysis

In this section an analysis of the Ruby dataset is provided, so each question posed in section 2 can be assessed. The numbering of the sub sections matches the numbering of the research questions in section 2 and each sub section analyzes the corresponding question.

[1] Ruby community's official gem hosting service. Available at http://rubygems.org
[2] The complete dataset can be downloaded from http://softwareecosystems.org

4.1 Elements

The units of analysis in the Ruby SECO are gems and developers, with the possible relationships among them. If a developer has a relationship with a gem, he is a developer of that specific gem. If a developer has a relationship with another developer they have worked together on a certain gem and if a gem has a relationship with another gem, they are dependent on one another. The dependency relationship among gems can have two different types, as can be seen in figure 1. Dependencies between gems can either be runtime dependencies or development dependencies. The first type is in place if a gem needs another gem in order to work properly for the end-user. The second type of dependency is relevant when a gem needs another gem only for development purposes, but not to work properly at runtime. An overview of all elements and their relationships can be found in figure 2.

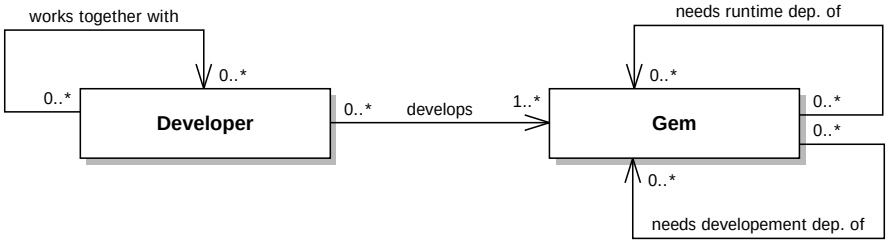


Fig. 2: Metamodel of interacting SECO elements

4.2 Element characteristics

Numerous characteristics are related to the SECO elements described above. For all gems we identified the following characteristics relevant for analysis, based on the information in the XML and the additional information available by downloading and examining the source code of the gem:

- **Name** - This is an unique identifier used to give a name to the particular gem, but also to be able to differentiate this gem from other gems. We use this characteristic only for identifying purposes.
- **Number of Downloads** - This number is an indication of the popularity of the gem. The more a gem is downloaded, the more popular it probably is. This popularity can for example be caused by the fact that this gem fulfils a core functionality and is used as a dependency by a lot of other gems.
- **Main Version** - This version number is a indicator of the maturity of the gem. Ruby uses sequence-based software versioning scheme in which a 0 as starting digit indicates a beta state and 1 or higher indicates a more mature

state of the gem (for example 1.0.6). We only looked at the starting digit of the versioning sequence. Please note that the digits are not comparable with each other; version 2 on one gem can indicate another level of maturity than version 2 on another gem, since developers can decide themselves when to increase their version number.
- **Lines of Code (LOC)** - The LOC is an important indicator of the effort that was put in developing a gem. Ruby gems have two types of code that are present in the source of the gem. Besides the code that provides the functionality for the gem, also test code in embedded in the gem source. For the sake of our analyses we looked at the total lines of code. This includes both the functionality LOC and the test LOC.
- **Size** - This characteristic indicates the amount of bytes a gem uses. The characteristic can also be used as an indicator of the total effort put in developing the gem, but this can be deceiving due to the fact that gems can exist out of more than only code (i.e. images).
- **Yahoo Hits** - The amount of hits that were generated by giving "$name of the gem + 'ruby gem"' as input to the Yahoo search engine. This is also an indicator of the gem popularity, since a popular gem is more likely to be named or linked on multiple locations on the web instead of only from one location.

The characteristics listed above will be used as variables in further sections of the paper.

4.3 Descriptives

First the entire ecosystem including all developers, gems and their relationships is visualized within one big graph using the network visualization software Gephi [17], as can be seen in figure 3[3]. The visualization gives a clear indication of the size of the Ruby ecosystem and also shows a lot of small projects and some larger interconnected projects. The structure of the ecosystem clearly indicates some sort of ecosystem 'core' of active developers, as can be seen by the interconnected 'stem' in the middle of the visualization.

Table 1: Ruby Ecosystem descriptives

Characteristic	Minimum	Maximum	Average	SD	Median
Downloads	3	377,251	1,159	10,710	123
Yahoo Hits	0	21,500,000	167,057	745,156	334
Size	0	25,851,477	76,362	422,091	16034
Lines of Code	0	427,736	2,059	8,544	513

The ecosystem consists of *4,784* developers, *10,046* gems and *13,103* relationships between them. A gem is developed by an average of *1.23* developers

[3] A high resolution version of the figure is available at http://softwareecosystems.org.

with a standard deviation of *0.725* and each developer in the ecosystem has developed an average of *2.53* gems with a standard deviation of *3.95*. A selection of some of the most important descriptives of the ecosystem is reported in table 1.

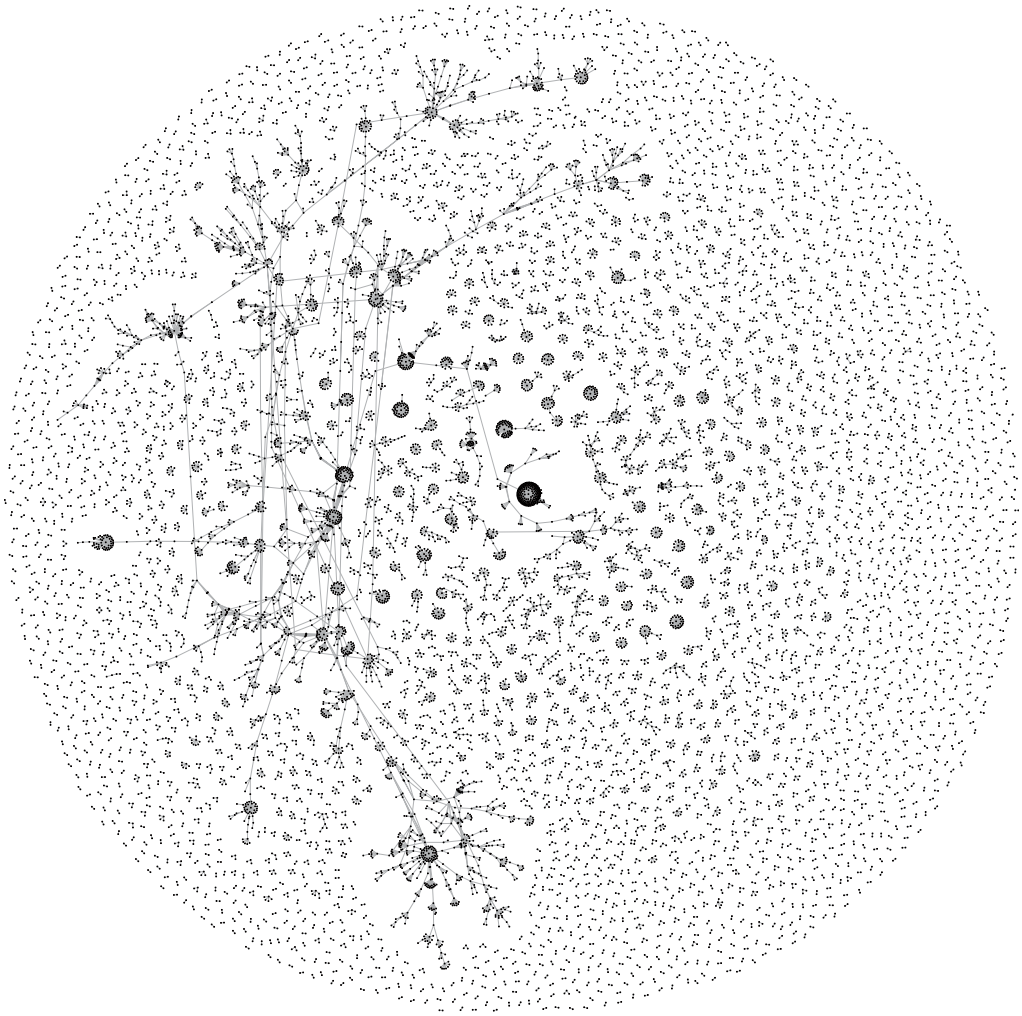
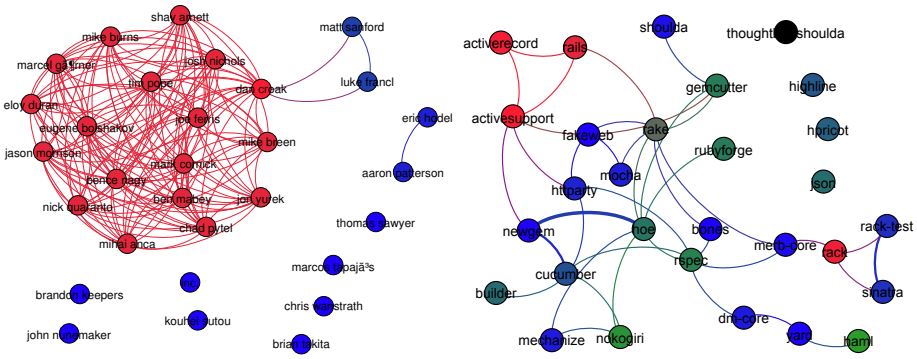


Fig. 3: Ruby Ecosystem Visualization.

In order to get insight in the Ruby ecosystem we first examined the boundaries of the ecosystem by analyzing the most popular gems and most active developers. We calculated the *eigenvector centrality* for all developers within the ecosystem to get a good overview of the level of importance of the developer. The eigenvector centrality is used within network analysis as a measure to indicate the importance of a node in the network [18]. For the sake of understandability

of the graph, only the top 30 developers in terms of degree [19] were included. Figure 4a shows the network of interacting developers with a range of degree between 15 and 30. Notable is the concentration of nodes on the left, which are all interconnected and indicate a dense interaction of a large number of most active developers. Figure 4b shows the top 30 of interdependent gems. This top is composed by selecting the gems that have the highest number of gems depending on them. We added up both runtime and development dependency for calculating this top 30. As can be seen, most crucial gems also have interdependencies among each other, indicating a strong network of important gems within the Ruby ecosystem.



(a) Top 30 of interacting developers (b) Top 30 of interdependent gems

Fig. 4: Overview of top 30 gems and developers

All of the developers shown in figure 4a are in the top 50 op highest eigenvector of centrality score and have an eigenvector centrality score of between 1 and approximately 0.1. The meaning of this will be explained in section 5.

4.4 Roles

Within the Ruby ecosystem several different roles for developers are identified based on the degree of the developer (number of other developers he cooperates with), the number of gems made and the popularity of the gems.

The first role identified is the **Lone Wolf**. The role of Lone Wolf is based on the role of a 'Niche Player' [5], Specific to this context it is defined as a developer who has developed gems that are of importance within the Ruby ecosystem, but has almost no connections with other developers. He produces useful content for the ecosystem, but works solitary. The importance of a gem can be determined in different ways. First the number of downloads of all gems a developer made combined are of importance, after this the amount of gems that are dependent on a gem play a role. Finally the amount of gems created by someone is of

importance in addressing the Lone Wolf role. Table 2 shows the top 5 of Lone Wolfs in the Ruby ecosystem. The ranking is based on the number of dependent gems as most important qualifier. These 5 developers mean a lot for the SECO, but are not related to any other developers.

Table 2: Lone Wolf Top 5

Developer	Number of downloads	Dependent gems	Number of gems
David Heinemeier Hansson	2,056,351	2146	13
Loren Segal	7.123	295	10
Bob Aman	53,198	268	10
Makoto Kuwata	73,874	180	17
Zed A. Shaw	114,546	164	9

The second role we identified is the role of **'Networker'.** The Networker role is based on the keystone role [5], but is defined for this specific case as someone who has a lot of developers he works with and also plays a large role in the SECO in terms of gem downloads. The top 30 of Networkers can be found in figure 4a. Besides Networkers we can also identify the so called **'One Day Flies'**. These developers have made one popular gem, but never made anything else. The criterion for being classified as a One Day Fly is: only one gem created with a version number starting with '0' and being in the top 5% of most number of downloads. A listing of the top 3 One Day Flies can be found in table 3.

Table 3: One Day Fly Top 3

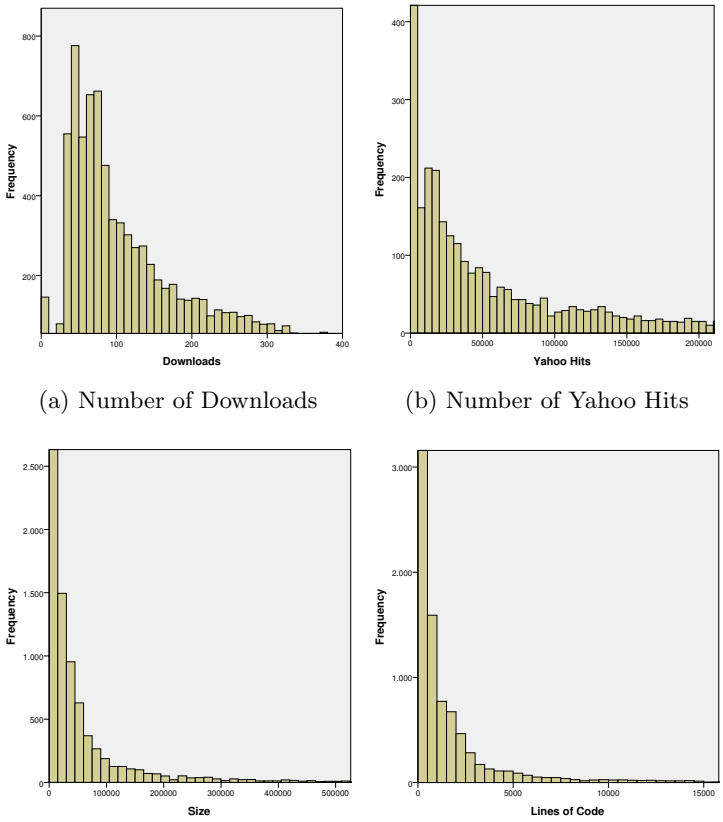
Developer	Gem	Version Number	Number of downloads
Wayne Meissner	FFI	0.6.1	46,222
Philip Ross	tzinfo	0.3.16	22,398
Benjamin Curtis	faker	0.3.18	13,522

All different roles identified each have different functions in the Ruby ecosystem, as will be further discussed in section 6.

5 Results

As section 4.1 showed, developers, gems and the relationships among them play a crucial role in the Ruby SECO. The characteristics playing a role in our analysis were listed and explained in section 4.2. For answering the question on SECO descriptives, table 1 shows the average number of downloads of a gem was *1,159*. This does not mean, in this case, that an average gem has approximately one

thousand downloads. The height of the Standard Deviation (SD) shows us that the individual number of downloads per gem differs significantly from the average. This conclusion is shown clearly in figure 5a, in which a high number of gems can be seen with less than 100 downloads. Around 90% of all gems has a number of downloads below the average, meaning that there is only a 10% of all gems responsible for the high number of downloads.



(a) Number of Downloads (b) Number of Yahoo Hits

(c) Size (d) Lines of Code

Fig. 5: Gem Characteristics Histograms

A similar situation can be seen when looking at the amount of hits gems got in the Yahoo search engine. The SD of *745,156* again is much higher than the average number of hits of *167,057*, meaning that there is a skewed distribution. Figure 5b[4] shows that the distribution of hits on Yahoo is indeed skewed and the

[4] One should note that the bar at '0' should actually be extended to a higher number, but is cropped due to readability.

number of gems that is has an amount of hits below the average is again around 90%. Since we use the amount of hits on Yahoo as an indication of popularity of a specific gem, this statistic says a lot about the distribution of popularity among gems. Figure 5c[4] and 5d[4] both show a similar distribution and indicate the same effect as the amount of downloads and the number of Yahoo hits did; a small part of the Ruby SECO in this case take care of the largest gems in terms of bytes and lines of code. Finally in section 4.4 we could identify three distinct roles within the SECO, each being of high importance for the SECO. The existence of these roles could indicate that there is not one 'holy grail' in governing a SECO, since there are different types of developers.

6 Discussion and Conclusion

The conclusions of this work are as follows: **(1)** the Ruby SECO consists of developers, gems and relationships and **(2)** developers within the SECO fulfil several distinctive roles, each of different value to the ecosystem. **(3)** Also, within the SECO most activity is caused only by a small part of the ecosystem. the top 90% of the open source components used in Ruby development has been developed by only 10% of the total number of open source contributors.. The value of this knowledge lies in deciding how to better manage or steer SECO governance. **(4)** Trying to lure additional developers to your ecosystem in order to expand your ecosystem may not be the best way of managing a SECO; motivating existing developers to work together more on existing gems is a better way to get a solid and healthy ecosystem. On the other hand, some of the most popular gems are developed by lone wolfs, so this conclusion should be investigated in more depth in future longitudinal research.

For the dataset we analyzed, we were dependent on the 'snapshot' of gems available on Github. This dependency caused us to not being able to analyze everything the way we would have wished. The developments and growth within the SECO could not be researched by the static dataset we had. Because of this, no strong conclusions can be made on how to stimulate growth or health. Also we only performed one case study, so our results can not be generalized to draw conclusion on FOSS SECOs in general. On the topic of analysis tools, the large size of the dataset resulted in very long computation times, making analysis sometimes difficult and cumbersome.

Due to the lack of longitudinal data it is impossible to speculate about the dynamics of the Ruby SECO. In the future, we plan to follow specific developers and clusters in the ecosystem. Methods such as structural break analysis can be applied to monitor SECOs, to early discover the specific events that have happened in the SECO. Presently an extensible tool is being developed that is able to mine several different repositories, so future repositories can be mined as well. The tool will be able to add the element 'time' to our analysis and can also give additional case studies rather easily. This will enable us to do more longitudinal research on the topic of software ecosystem evolution in the future and generate more generalizable results on software ecosystems in general.

Acknowledgements - The authors would like to thank Bernard Verhoeven for supplying us with the data used for our analyses and his useful suggestions.

References

1. Xu, L., Brinkkemper, S.: Concepts of product software. European Journal of Information Systems **16**(5) (2007) 531–541
2. Messerschmitt, D., Szyperski, C.: Software ecosystem: understanding an indispensable technology and industry. The MIT Press (2005)
3. Jansen, S., Finkelstein, A., Brinkkemper, S.: A sense of community: A research agenda for software ecosystems. 2009 31st International Conference on Software Engineering Companion Volume (2009) 187–190
4. Popp, K.M., Meyer, R.: Profit from Software Ecosystems. Books on Demand GmbH (2010)
5. Iansiti, M., Levien, R.: The keystone advantage: what the new dynamics of business ecosystems mean for strategy, innovation, and sustainability. Harvard Business Press (2004)
6. Feller, J., Fitzgerald, B.: Understanding open source software development. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2002)
7. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Understanding free/open source software development processes. Software Process: Improvement and Practice **11**(2) (2006) 95–105
8. Madey, G., Freeh, V., Tynan, R.: Modeling the free/open source software community: A quantitative investigation. Free/Open Source Software Development (2004) 203–220
9. Yin, R.K.: Applications of case study research: second edition. SAGE (2003)
10. Kagdi, H., Collard, M., Maletic, J.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice **19**(2) (2007) 77–131
11. Ross, S.: Introductory statistics. Academic Press (2005)
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-wesley Reading, MA (1995)
13. Bächle, M., Kirchberg, P.: Ruby on Rails. IEEE software (2007) 105–108
14. Bird, C., Rigby, P., Barr, E., Hamilton, D., German, D., Devanbu, P.: The promises and perils of mining git. In: 6th IEEE International Working Conference on Mining Software Repositories. (2009) 1–10
15. Rosen, L.: Open source licensing: Software freedom and intellectual property law. Prentice Hall PTR Upper Saddle River, NJ, USA (2004)
16. W3C: Extensible markup language (xml) 1.0 (fifth edition) (November 2008)
17. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks. In: International AAAI Conference on Weblogs and Social Media. (2009)
18. Bonacich, P.: Some unique properties of eigenvector centrality. Social Networks **29**(4) (2007) 555 – 564
19. West, D.: Introduction to graph theory. Prentice Hall (2001)