Carina F. Alves, Geir K. Hanssen, Jan Bosch, Slinger Jansen (Eds.)

Proceedings of

5th International Workshop on Software Ecosystems (IWSECO 2013)

Workshops hosted by 4th International Conference on Software Business (ICSOB 2013) in Potsdam, Germany, June 11, 2013

Addresses of the editors:

Carina F. Alves
Universidade Federal de Pernambuco
Cid. Unversitária s/n,
CEP 50.740-560, Brazil
cfa@cin.ufpe.br

Geir K. Hanssen
SINTEF ICT
Box 4760 Sluppen,
N-7465 Trondheim,
Norway
ghanssen@sintef.no

Jan Bosch
Chalmers University of Technology
SE-412 96 Gothenburg,
Sweden
jan@janbosch.com

Slinger Jansen
Utrecht University
PO Box 80.089
3508 TB Utrecht,
The Netherlands
slinger@slingerjansen.nl

Contents

Towards the Analysis of Software Projects Dependencies: An Exploratory Visual Study of Software Ecosystems

Francisco W. Santana and Cláudia M. L. Werner

PESC - COPPE/UFRJ,
Universidade Federal do Rio de Janeiro (UFRJ),
Caixa Postal 68511 – CEP 21945-970 – Rio de Janeiro, RJ
{fwsantana,werner}@cos.ufrj.br

Abstract. Software systems are rarely developed in isolation. The development of current complex software systems often makes extensive use of components previously developed or acquired from external suppliers. Nevertheless, research on software systems has traditionally considered each system as an isolated and self-contained project. Such characteristic limits the quality of observations on studies as they do not consider the software ecosystem in which the project is inserted. In this paper we present an ongoing work that aims to enable the analysis of software ecosystems from both technical and sociotechnical perspectives. The novelty of our approach lies in the usage of interactive visualizations to facilitate uncovering relationships among software projects within an ecosystem.

Key words: Software Engineering, Software Dependencies, Software Ecosystems, Software Visualization, Mining Software Repositories

1 Introduction

Software systems are increasingly dependent on other software systems. The need to cope with an increasingly competitive market and demanding customers leaves no room for each new complex software system to be completely developed from scratch. Software industry has therefore recurred to software reuse in order to achieve its goals, making extensive use of solutions previously developed or acquired from external suppliers and incorporating them within its products.
Although the usage of software components can offer several advantages to the development of software projects, such as reduced development time, higher productivity and reliability, a poor choice of components can undermine the quality of the developed product [1]. However, most of the studies regarding effects and impacts related to software components and general dependencies consider each project as an isolated and self-contained product [2][3]. This perspective ignores the relationships between a software project and its dependencies, and also the relationships among software communities involved in the development of these projects, i.e., the software ecosystem in which the project is inserted.

Software ecosystems (SECOs) have become a subject of great interest for both industry and academic communities, motivated by this holistic view of software development that goes beyond exploring a single project and its own entities. We define software ecosystems as a set of projects that are interdependent from both technical and sociotechnical perspectives, that is, project artifacts, developers and supporting communities, linked to other projects by either reuse dependencies or common contributors of development communities. As suggested by related work [4], we believe that this view is important to better comprehend software projects' development characteristics, e.g., uncover details about the impacts of the chosen reused components in a specific software project.
This paper aims to explore software component dependencies from an ecosystem point of view, under the hypothesis that characteristics of reused components impact on the developed system that reuses them. The novelty of our approach resides on the employment of visualizations to observe and explore software dependencies, uncovering relationships and patterns among interrelated projects. Our motivation is grounded on our perception that few works [4][5] on literature tries to comprehend the impacts of components within different software projects from both technical and sociotechnical perspectives, constituting a new scenario for studies on software engineering.
The rest of this paper is organized as follows. In Section 2, we introduce concepts of Software Ecosystems, detailing our definitions and comparing them to those of other works. Section 3 presents an overview of existing work involving visualization of software ecosystems. Section 4 describes our approach, including notions that comprise its foundations and visualization techniques. Section 5 presents a proof of concept conducted to demonstrate the feasibility of our approach. Finally, Section 6 presents our final remarks and plans for future work.

2 Software Ecosystems

As stated by Lungu et al. "no software project is an island" [6]. The development of current software systems are rarely conducted in isolation, with solutions being redesigned and redeveloped. In this sense, the development of a software system by the composition of reusable components establishes relationships among various software projects and development communities, forming a genuine ecosystem that guides and constraints the development of a software system. Software ecosystems have become an increasingly popular subject, with several research works being conducted to explore and understand relationships among the various entities involved in software systems development. Nevertheless, the study of SECOs is still a novel area and there is no consensus on the definition of what constitutes a software ecosystem. Based on a recent systematic mapping study [7], the term was coined in 2003 as a collection of software products that have some given degree of symbiotic relationships [8]. More recent work however tends to identify as a remarkable feature of a SECO the idea of a common market or platform, inserting the projects in relationships of interest that are somehow business-oriented. For example, Boucharas et al. [9] define

SECOs as a set of actors functioning as a unit and interacting with a shared market for software and services, while Bosch [10] sees SECOs as a set of software systems that enables, supports and automates activities and transactions in the associated social or business ecosystems.
Albeit the aspiration to reach a bigger share on a particular market segment can be observed in software industry and therefore on many software products, inter-projects relationships may appear for other reasons than business-oriented, ranging from convenience to reuse a previously developed solution to simple interest in adopting a particular technology. These relationships also establish connections between projects, forming bonds that, when viewed in a broad perspective, can be seen as a software ecosystem. Based on this reasoning, we adopted a broad definition for SECOs, independent of business features and also aligned to other recent works [11][12].
There are many challenges and unanswered questions regarding SECOs, particularly from perspectives such as software maintenance and evolution. One such challenge involves the risks that a software project faces while entering a SECO. Even though reusing software components with assured quality can provide benefits to software projects, a software ecosystem can also impose obstacles to the development of a software project when the project is made upon components with low technical quality. Also, these challenges cannot be seen from a strictly technical point of view, as both technical and social factors play an important role on the process of software development [2] [3] [13].

3 Existing Work

Though recent studies have explored characteristics and behaviors of SECOs, few approaches have been concerned with ways to visualize and analyze SECOs relationships. Among the studies on this topic, Pérez et al. [4] developed a tool for visualization and analysis of ecosystems named SECONDA, a tool that collects data from GIT repositories, computes software metrics from source code artifacts and presents visualizations of the processed data from several projects simultaneously, allowing the exploration of SECOs. Another important work presents the Small Project Observatory (SPO) [6], which also collects data and presents visualizations from multiple projects in an integrated manner, offering users two different perspectives to explore SECOs, with focus on either ecosystems' projects or developers' communities.
Both SECONDA and SPO offer mechanisms for data extracting, metrics evaluation and visualization of projects within SECOs for visual analysis. However, these tools have limitations related to data collection, being able to only process a so-called "super-repository" that contains all projects of a SECO, regardless of the fact that the data of all SECOs' projects are not necessarily stored in one common repository, limiting the analysis that can be done with these tools and posing as a threat to validity for these works. Furthermore, while these tools offer mechanisms for researchers to explore SECOs from both technical and sociotechnical perspectives, they do not provide means to explore how the

relationships evolved over time.
Finally, there are other important but less related works that show charts and indicators of ecosystems and relate to our proposal by providing results that can assist us in formulating hypotheses, choosing metrics and constructing visualizations that are applicable to analyze SECOs. Among this category of studies, stands out the one described in [13] that analyzed the Evince document viewer and presented a serie of views that could also be applied to software ecosystems, and the one detailed in [14] that found closer relationships between developers' communities of KDE and Gnome than Apache's due to technical proximity between the former projects.

4 Our Approach

Our approach consists on the usage of information visualization techniques to depict a SECO, applied over software projects repositories' data extracted using mining software repositories (MSR) methods. Our proposal features three main steps: data extracting, data processing, and SECO visualization. On the following subsections we describe each step and introduce key concepts to enable a better comprehension of our proposal.

4.1 Data Extracting

The first step of our proposal consists of obtaining data from version control systems and issue trackers of software projects that share technical dependencies, and are therefore members of a common ecosystem under our definition. Software repositories such as VCS and issue tracking systems are widely used by software developers on both commercial and open source environments, and contain a plethora of available data about the underlying software projects and associated development process that can be mined to explore and investigate evidences about software development. Access to these software repositories is made using native Subversion protocol for VCS connection, while issue trackers data are collected using webcrawlers. Our proposal's data extraction is not limited to analyzing only one repository, being able to collect data from projects stored in different repositories to compose an ecosystem.
Following the data collection of a software project, we automatically identify a list of software dependencies from build managers and dependency managers' artifacts, such as Maven's pom.xml or Ivy's ivy.xml. These tools are also extensively used by software development communities and assist the development process, offering features such as dependencies browsing and managing conflicts. The identification of software dependencies may lead to data extraction of other software projects, and despite the fact that dependencies identification can be performed automatically, data extraction needs to be conducted manually because repositories addresses and their access credentials are not specified in the

build managers' artifacts. Source code analysis was discarded as a way to establish links between projects since artifacts path or contents is not reliable to infer which project the artifact belongs to.

4.2 Data Processing

The Data processing phase has two distinct objectives, beginning with the identification of dependencies between the collected ecosystems entities, these entities being developers, actors on issue tracker communities, and software projects' artifacts and metrics computation. The dependencies among the SECOs entities are processed using three different techniques, namely logical dependencies, static dependencies and coordination requirements. Static dependencies arise from procedures and methods calls between projects' classes or compilation units, while in contrast logical dependencies [15] are computed between artifacts that have evolved and been modified many times together, even though these artifacts have no explicit connection. Finally, as software development is a typical example of collaborative work, the notion of coordination requirements proposed by Cataldo and colleagues [16] appears as a way to obtain the set of individuals a developer should coordinate his/her work with (or at least be aware of) based on their tasks level of interdependency, measured based on which artifacts each developer modified and the logical dependencies these artifacts shared. Building upon the identification of dependencies, the metrics evaluation phase has the objective to calculate SECOs software metrics to build indicators of a SECO feature (e.g., its health). Despite our intentions to evaluate metrics and construct indicators for SECOs features, there is a lack of formal definition for metrics on SECOs given the novelty of the discipline, and thus in our initial exploratory context we have employed but visual means to infer projects cohesion and coupling.

4.3 SECO Visualizations

Information visualization techniques have been successfully employed by researchers trying to understand software system features, especially those related to software evolution, maintenance and monitoring [17]. As many SECOs features remain unknown, we believe that visualizations offer a sound starting point to explore the relationships of related software projects, and developed two visualizations to observe different characteristics of SECOs, namely communities and technical views.
Higher resolution images of the visualizations presented on next subsections are available at http://lab3d.coppe.ufrj.br/index.php/projetos/visecos.

Communities View This view (Figure 1) focuses on the analysis of how the different communities of a SECO interact, i.e., how each project member of the SECO contributes to other projects from a sociotechnical point of view. The main usage scenario of this visualization is to identify the degree of participation and contributions between different projects of the same SECO, naively

suggesting the health of the whole SECO community.
To represent these interactions within SECOs' communities we chose to use graphs, since it is regarded as an intuitive way to display dependencies. Based on data from both VCS repositories and issue trackers, we constructed two graphs at different levels of granularity. On the coarse grained graph, each project is represented by a vertex with a unique color and has two child vertices, representing the VCS developers' communities and issue trackers contributors. Each of these communities' vertices can then be linked by edges to vertices of other projects, meaning that there are collaborators that act on both communities. The fine grained graph in turn depicts interaction between the communities actors more explicitly based on both Coordination Requirements network and discussions on the same issue by different contributors.

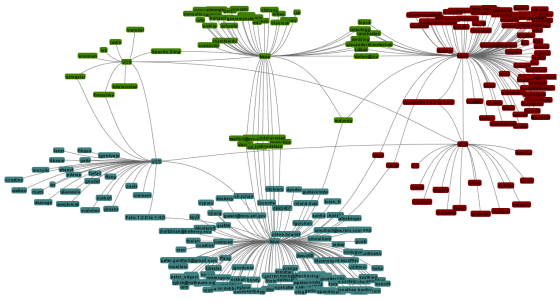


Fig. 1: Communities View, featuring Apache Sling (green), Apache Jackrabbit (red) and Apache Felix (blue) projects.

The user can interact with the visualization by applying various filters, enabling to focus on the relationships that are of interest to his/her analysis. The filters currently implemented allow: i) hiding nodes from a specific project or archive (VCS or issue tracker), enabling to focus on a selected set of projects of the SECO; ii) coloring the nodes based on how many different projects a contributor participates; iii) change the graphs granularity; iv) and select a specific time-span to focus on, considering only contributions made within this period.

Technical View This view focus on the analysis of how technically interdependent of each other the different projects within a SECO are, i.e., how the various artifacts/modules of one project make use of methods and/or objects declared on other projects' artifacts/modules. This visualization is employed to observe the most important artifacts of the ecosystems from a reuse perspective, enabling the visual identification of various characteristics such as the assets that are more central to the SECO (i.e., that various other artifacts depend upon) or those that depend upon artifacts supplied by several other projects.
The technical view also makes use of graphs to evidence the links between artifacts (Figure 2), displaying the processed static dependencies. Technical view's

graph also offers filters to conduct analysis over fine or coarse grains, since the amount of dependencies between the various projects can prove to be too dense to interpret. In the fine-grained analysis all vertices are artifacts identified by the project's color, and edges represent dependencies between the artifacts, whilst on coarse-grain the vertices represent the projects modules obtained from the artifacts' package declaration, naively hinting at the components of each project.

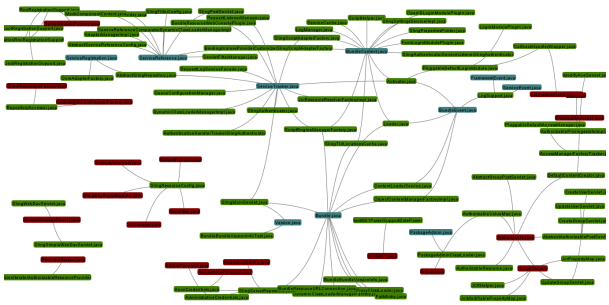


Fig. 2: Technical View presenting dependencies between Apache Slings artifacts (green) and Apache Felix (blue) and Apache Jackrabbit (red) projects.

To adequately observe the SECO technical dependencies, users can interact with the visualization likewise the communities view, being able to change the granularity of dependencies and show/hide artifacts of specified projects.

5 Proof of Concept

To demonstrate the feasibility of our approach, we conducted an exploration of the relationships of a SECO in the form of a proof of concept. The main objective was to highlight the usage of the visualizations we proposed in a real case, using open source projects. Our expectations were that the visualizations would help to identify how dense the technical and sociotechnical relationships between a software project and other projects that it depended upon were.

5.1 Project Description

In order to observe the relationships of a SECO, we required a software project that satisfied three requirements: i) was hosted on Subversion (SVN) repository that we had read access; ii) reported issues on either JIRA or Bugzilla issue tracking system; and iii) listed its dependencies in structured artifacts (e.g., Apache Maven's pom.xml or Apache Ivy's ivy.xml). The first two requirements are due to limitations of our supporting tools, which can only collect data from the specified repositories. The latter is given due to the technique we used to infer software project dependencies, using the specified artifacts to automatically

gather this information.
Given our intentions and limitations, we chose to observe Apache Sling's ecosystem. Apache Sling is a large open source web framework designed to store and manage content over a Java Content Repository (JCR) specification developed by Apache Software Foundation (ASF), being one of its top-level projects since 2009. Apache Sling also matches all our established requirements: it is hosted on a public SVN repository, uses JIRA as issue tracking system and Maven as dependency manager. Furthermore, the project was handily picked due to our previous knowledge of its dependencies with other large ASF's top-level projects Jackrabbit and Felix.

5.2 Supporting Tools

Extracting relevant data depicting how software systems were developed is a complex task. The sheer amount of data can easily overwhelm researchers if appropriate tools and methods are not applied, misleading them to erroneously identify inexistent patterns and features. We made use of two tools to assist us on reaching our goals, namely XFlow and JDX.
XFlow [18] is an extensible open source tool that enables empirical software evolution analyses from both technical and sociotechnical perspectives. We used XFlow to extract and process projects' data, evaluate metrics of the selected projects, and also adapted the tool to extract software projects' dependencies and present the visualizations described on the previous subsection.
Java Dependency eXtractor (JDX) is a library developed to identify static dependencies from java source code artifacts, being able to compute call-graphs using Eclipse IDE compiler's JDT Core to handle source code in plain form. We used JDX to compute static dependencies between source code artifacts.

5.3 Analysis Procedure

We analyzed one specific release of Apache Sling, developed from May 14th, 2009 to April 18th, 2011. The information about the release dates was obtained by browsing the project's mailing lists and looking for release announcements. In this period, the project was actively developed by a group of 7 people. Apache Sling is composed of several modules within the same repository, and our data collect procedure was configured to extract data from all these modules' trunk directories, resulting in 830 commits found. We then proceeded with the data gathering by accessing and collecting data from Sling's JIRA, collecting 318 issues reported and/or resolved by a supporting community of 51 people.
After this initial data collect we used XFlow to analyze Sling's dependencies and choose two related projects to consider on our analysis: Apache Felix (release 1.2.0 to 1.4.0) and Apache Jackrabbit (release 1.5.0 to 1.6.0). Apache Felix is an open source implementation of OSGi R4 Service Platform and other OSGi-related technologies, being widely used by well-known projects such as Glassfish application server and the Netbeans IDE; while Jackrabbit is an implementation of Content Repository for Java (JCR) specification, and provides content

management services. We analogously collected data from these selected projects VCS trunks and issue trackers, ending up with a small portion of Sling's ecosystem for study.
For the data processing phase, we obtained the source bundle of analyzed projects on their official site to identify static dependencies between the artifacts and modules, while logical dependencies were identified using data extracted from VCS and issue trackers by XFlow. We then proceeded to generate visualizations and analyzed our results, discussed on the next subsection.

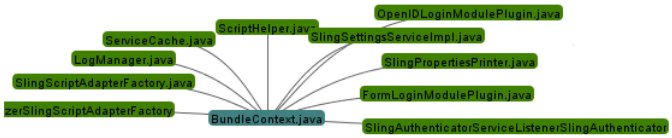
5.4 Results

Our exploratory analysis was able to successfully depict interaction among projects under our SECO definition (Figure 1 and Figure 2). This initial exploitation was a necessary step to justify further studies, since we couldn't predict how intense would be the interaction among communities of related software projects. Also while our visualizations may not scale to display a large amount of data, we believe that filtering graph nodes using metrics such as centrality and betweenness centrality on future studies can reduce these problems.
Besides verifying the feasibility of our approach, we observed interesting features on the analyzed SECO. The discussion of these findings has been divided in technical and sociotechnical perspectives.

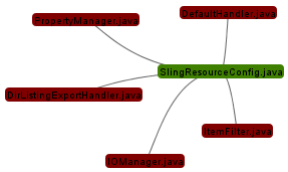
Technical Discussion For the technical perspective analysis we focused on the dependencies between Apache Sling and both Apache Felix and Jackrabbits, excluding dependencies between the latter projects. We did that since the graph that resulted when considering all projects was too dense and proved difficult to represent in a legible figure and was beyond our exploratory context.
Considering this limitation, we have analyzed the reused artifacts and the technical view suggests that there are some critical artifacts to the maintenance of Apache Sling that are provided by other projects. We did not find any class that depended upon artifacts from two projects, and identified two interesting patterns on the assets reused: i) a large set of classes that depends on a single class of another project (Figure 3(a)); and ii) a class that depends on various artifacts of other projects (Figure 3(b)). These patterns suggest how prepared to reuse these components were, implying on the reliability of the modules that depend on them. In the first pattern, for instance, a defect on the reused artifact could be propagated to several artifacts that reuse it; while the second one indicates that even small changes in a class could require knowledge of several other classes from other projects.

Sociotechnical Discussion From a sociotechnical perspective, we observed that all projects of the SECO were related (Figure 1). We identified that most of the interactions between the communities involved on the analyzed period has occurred via issue tracking systems, in which 8 collaborators from Apache Sling's issue tracker contributed to project Apache Jackrabbit's issue tracker and vice-versa, 7 between Apache Sling and Apache Felix, and 3 between Apache Felix

and Apache Jackrabbit. We also note that there is one contributor that bridges all the projects, contributing to all three issue tracker. Looking at the VCS communities, we observed that few developers had participated on two projects on the same period (3 between Apache Sling and Apache Felix), while none had participated in all three.



(a)



(b)

Fig. 3: Patterns identified on technical visualization.

Based on these exploratory findings, we conclude that there have been considerable contributions between all the projects of the SECO and reaffirm the importance of sociotechnical analysis on software engineering studies. This initial analysis also provides basis to the conception of an experimental hypothesis that needs further studies to be answered, regarding topics such as the role of those contributors who participate on various projects inside a SECO, or the identification of features that facilitate or hinder the contribution towards a specific project from external developers that reuse its assets.

5.5 Threats to Validity

Our study is characterized as a proof of concept, and thus has not the same rigor as a formal experiment. Nevertheless, we found interesting results that must be interpreted considering the following validity threats.

Construct Validity. The relationships between developers that we have found may not be totally accurate as it's common to find some contributors who do not have permission to commit to the project's VCS and have their contributions carried out by others developers.

Internal Validity. Our approach is based on the hypothesis that features of a project's dependencies impact the project that uses them as components, but we have not extensively verified the correlation between the projects. Considering this, we cannot determine that the characteristics we observed actually come

from the related projects or are derived from unobserved attributes.
External Validity. As we analyzed few projects, all of them belonging to the same organization (ASF), we do not claim that these results remain valid for other projects and in different development contexts.
Conclusion Validity. We did not conduct any statistical analysis in order to validate our findings, making use of purely visual methods to perform our analysis. While this could be interpreted as a serious threat to our findings, given the lack of related work with similar objectives we decided to first conduct analyses without statistical rigor to explore the degree of interdependencies between projects and general interaction of SECO communities.

6 Final Remarks and Future Work

This paper presented an approach to analyze SECOs from a technical and sociotechnical perspective, focusing on projects that share reuse dependencies. Our exploratory study matched our expectations that the different projects interact beyond the technical level, and this motivated us to further inspect how features of a reusable component affect software projects and the contributors' communities that reuse them.
We remark that this paper presents an initial effort to explore SECOs and expect to expand this proposal under the following two major aspects:
SECO Metrics: given that software ecosystems form a recent subject for Software Engineering studies, there is a lack of papers that list formal metrics and indicators applicable for SECOs. We intend to conduct a systematic mapping of studies to identify metrics used by recent works on SECOs context, gathering a list of metrics that will enable us to further investigate SECOs.
Expand Research Perspective: our initiative is part of a greater context of SECOs exploration, with other researchers interested in the investigation of IT and Governance aspects and the proposal of a framework for modeling and managing SECOs [19]. In the future we expect to integrate our solutions and provide more complete studies on SECOs.

Acknowledgments. Francisco Santana receives individual grant from CAPES, and Cláudia Werner from CNPq.

References

1. Jiang, L., Carley, K., Bigrigg, M., Eberlein, A., Galster, M.: The impact of component interconnections on software quality: A network analysis approach. In: Systems, Man, and Cybernetics (SMC'12). (2012) 1865–1872
2. De Souza, C.R.B.: On the relationship between software dependencies and coordination: field studies and tool support. PhD thesis, Long Beach, CA, USA (2005)
3. Cataldo, M., Mockus, A., Roberts, J.A., Herbsleb, J.D.: Software dependencies, work dependencies, and their impact on failures. IEEE Transactions on Software Engineering **35**(6) (2009) 864–878

4. Pérez, J., Deshayes, R., Goeminne, M., Mens, T.: Seconda: Software ecosystem analysis dashboard. In: Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on. (2012) 527–530
5. Goeminne, M., Mens, T.: A framework for analysing and visualising open source software ecosystems. In: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE). IWPSE-EVOL '10, New York, NY, USA, ACM (2010) 42–47
6. Lungu, M., Lanza, M., Gîrba, T., Robbes, R.: The small project observatory: Visualizing software ecosystems. Sci. Comput. Program. **75** (April 2010) 264–275
7. Barbosa, O., Santos, R., Alves, C., Werner, C., Jansen, S.: A systematic mapping study on software ecosystems through a three-dimensional perspective. In Jansen, S., Brinkkemper, S., Cusumano, M., eds.: Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry. Edward Elgar Publishing, Cheltenham, UK, and Northampton, MA, USA (2013)
8. Messerschmitt, D.G., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press, Cambridge, MA, USA (2003)
9. Boucharas, V., Jansen, S., Brinkkemper, S.: Formalizing software ecosystem modeling. In: Proceedings of the 1st international workshop on Open component ecosystems. IWOCE '09, New York, NY, USA, ACM (2009) 41–50
10. Bosch, J.: From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference. SPLC '09, Pittsburgh, PA, USA, Carnegie Mellon University (2009) 111–119
11. Cataldo, M., Herbsleb, J.D.: Architecting in software ecosystems: interface translucence as an enabler for scalable collaboration. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ECSA '10, New York, NY, USA, ACM (2010) 65–72
12. Bosch, J., Bosch-Sijtsema, P.: From integration to composition: On the impact of software product lines, global development and ecosystems. J. Syst. Softw. **83**(1) (January 2010) 67–76
13. Mens, T., Goeminne, M.: Analysing the evolution of social aspects of open source software ecosystems. In: Proceedings of the Third International Workshop on Software Ecosystems. IWSECO '11 (2011) 1–14
14. Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M., Herraiz, I.: 3. In: Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects. Volume 1. Information Resources Management Association (2009) 28–50
15. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Software Maintenance, 1998. Proceedings., International Conference on. (1998) 190–198
16. Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., Carley, K.M.: Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. CSCW '06, New York, NY, USA, ACM (2006) 353–362
17. Diehl, S.: Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
18. Santana, F.W., Oliva, G.A., de Souza, C.R.B., Gerosa, M.A.: Xflow: An extensible tool for empirical analysis of software systems evolution. In: Proceedings of the VIII Experimental Software Engineering Latin American Workshop. ESELAW '10, New York, NY, USA, ACM (2010) 353–362
19. Santos, R.P., Werner, C.M.L.: Reuseecos: An approach to support global software development through software ecosystems. In: ICGSE Workshops. (2012) 60–65

On Clusters in Open Source Ecosystems

Shaheen Syed and Slinger Jansen

Department of Information and Computer Sciences, Utrecht University
Princetonplein 5, 3508 TB Utrecht, the Netherlands
{s.a.s.syed, slingerjansen}@uu.nl

Abstract. This paper seeks to find characteristics of relationships between developers within various clusters of FLOSS ecosystems. We have mined the repository of the open source programming language Ruby, and linked developers and projects on the basis of collaboration. We used Social Network Analysis, and more specifically, the concept of modularity to expose underlying clusters or sub-communities. A survey was constructed to aid in the qualitative part of this research. The data shows that Ruby's ecosystem consist mostly of single developers who work independently. Developers within clusters of a few developers often have personal relationships formed through friendship, work and the open source community. Personal relationships formed through the open source community grow as clusters consist of more developers. Developers in clusters with large number of developers are often unaware of friend-of-friend relationships. Project administrators, however, fail to list developers that contribute through pull/request issues as authors, making data on Ruby's repository incomplete.

Keywords: Software Ecosystems, Clusters, Ruby, FLOSS

1 Introduction

Free/Libre Open Source Software (FLOSS) developers have to work collaboratively in order to maintain, improve or extend their software. Especially when their collaboration is underpinned by a common technology or market, and operate through the exchange of information, resources and artifacts, Software Ecosystems (SECOs) start to emerge [1]. Jansen et al. state that getting insights into a SECO requires the quantification and measurement of specific SECO characteristics. Possible characteristics are the number of sub-communities, the reciprocity of the ecosystem or the out degree of keystone actors. Moreover, understanding SECO characteristics can aid in maximizing its profitability, and more concretely, SECO orchestrators can develop strategies to keep a SECO vibrant and profitable for other organizations in the SECO.

This paper addresses the characteristics of sub-communities or clusters within the Ruby SECO. Clusters are defined as sets of nodes which are connected together by some path, that is, there exists a sequence of links that can be traversed to go from one node to any other node. There exist, however, no paths between two nodes in different clusters [3]. As SECOs can be seen as large

social networks, where actors (developers) are connected through relationships (collaborations) that hold them together [4], numerous clusters are present.

The study of SECOs, or social networks, can be conducted by applying Social Network Analysis (SNA). Several studies have applied SNA in the context of open source. For example, Gao, Freeh and Madey used SNA to study core developer networks on SourceForge.net hosted projects [5]. Lopez-Fernandez et al. used SNA to analyze source code repositories of open source projects [6]. Madey et al. applied SNA to model the open source network as collaborative networks and studied the growth of these networks [7]. Previous studies are, however, focused on the developer or project level, and not specifically on the cluster or sub-community level. Concretely put, this paper uses SNA to reveal underlying clusters within the Ruby SECO and strives to define the characteristics of relationships between developers. The ability to find and analyze such clusters can provide invaluable help in understanding and visualizing the structure of networks and thus understanding the SECO. Several algorithms exist to find sub-communities or clusters within networks [8–10]. Popular algorithms for large networks are based on the concept of modularity, which take the structure of a network and decompose it into modular communities, i.e. sub-communities or clusters. It looks for dense connections (relations) within groups, and sparse connections between them. Furthermore, it requires less computational complexity in contrast to other methods, e.g. graph partitioning. Our study uses the Louvain method [11], a modularity algorithm that has successfully been applied to find sub-communities in large networks, such as Twitter, LinkedIn, and Flickr. To extend our analysis, a survey is constructed to aid in the qualitative part of relationships between developers. Hence, Ruby developers were asked numerous questions related to other developers that were clustered together. Doing so enabled us to classify qualitative data by cluster sizes.

The remainder of the paper is structured as follows: Section 2 elaborates on the research method and provides insights into the various steps undertaken from data extraction to cluster identification. Furthermore, it elaborates on the survey and how it was constructed. Section 3 discusses the first findings of the data that were extracted from the Ruby repository and provides an overview of distributions that define the Ruby ecosystem. Section 4 makes a first classification of survey results based on various cluster sizes. Finally, Section 5 concludes our classification of results and provides improvements for future work.

2 Research Design

Our study has taken as object of study the Ruby open source SECO. Ruby is an object oriented programming language designed and developed in 1995 by Yukihiro Matsumoto in Japan. The syntax is influenced by Perl and Smalltalk features and is similar in many respects. It was created to balance functional programming with imperative programming. Ruby's most popular framework, named Ruby on Rails, is an open source web framework that enables developers to create web applications more easily. These applications, best described

as utilities or libraries, are called *gems* and are hosted on Ruby's repository Rubygems.org. Several gems can be combined to create specific features or extend existing work.

2.1 Research Question

This paper is an extension of our previous explorative study, in which we presented elements, characteristics, descriptives, roles, cliques and relationships of the Ruby ecosystem [12]. Our previous work was quantitative in nature and lacked essential qualitative information. We replicated the study and focused on relationships between developers within clusters. Thus, the main research question answered in this paper is: *What are the defining characteristics of relationships between developers within clusters of FLOSS ecosystems?* To answer this question, we looked at relationships between developers within the Ruby SECO of various cluster sizes with respect to the number of developers residing within them.

Developers are connected by strong or weak ties. For example, developers have strong ties if they are friends, classmates or working at the same company (i.e. they know each other personally to some extent), and have coded together to produce a working gem. Weak ties, by contrast, involve infrequent and limited interaction. For example, developers are connected together through a friend-of-friend relationship, that is, developer a has worked with developer b, who in turn has worked with developer c, developer a and c are now connected through developer b. Our study further looks at these friend-of-friend relationships and to what extent they constitute strong or weak ties.

Strong ties are important because it improves the exchange of information with other strong ties and reinforce companionship and support. The study of strong ties can help us understand the structure and local information of a social group [3]. Weak ties bring us more new opportunities and resources which cannot be obtained from close relations [13]. The study of weak ties provides a global view of the community as a whole.

2.2 Data Gathering

We collected our data from Rubygems.org as it provided us with an API to mine the data. Because integral collection was not possible, we developed several scripts to extract the data one by one from the API's XML output. The data was then stored in a relational database. The API of Rubygems.org provided us with the following data on each gem:

1. Gem name
2. Total downloads
3. Current version
4. Downloads current version
5. Authors
6. Info
7. Project uri
8. Gem uri
9. Homepage uri
10. Wiki uri
11. Documentation uri
12. Mailing list uri

13. Source code uri
14. Bug tracker uri
15. Dependencies

As our research was targeted at developers (authors), we distilled this information from the complete dataset. Unfortunately, Rubygems.org stored author information as a string, causing problems when multiple authors have collaborated on a gem. Several formats for entering multiple authors were used by project administrators, such as "Peter and John", "Peter & John", "Peter, John". This information was decoupled using several SQL statements which ultimately resulted in database records with single entities. Each gem was given a unique id and was linked to one or multiple authors. Data extraction was performed on May 7th 2012, providing us with a snapshot of the Ruby ecosystem as it was then.

2.3 Cluster Identification

After data collection we visualized the data using *Gephi* [14], which is a free to use application for social network analysis. Furthermore, Gephi is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs.

A common property within networks is *community structure*. It is the division of network nodes into groups within which the connections are dense, but between which they are sparse [9]. Newman and Girvan defined the measure of *modularity* to quantify the strengths of communities by using the denseness and sparsity of the group's inner and outer connections. Their modularity measure is a scalar value between -1 and 1, where 1 indicates networks with clusters that are disconnected from each other, i.e. no nodes overlap two clusters. Their algorithm is especially useful for undirected and unweighted graphs to reveal underlying clusters.

To identify clusters within the Ruby ecosystem we used the algorithm proposed by Blondel, Guillaume, Lambiotte and Lefebvre [11], known as the Louvain algorithm. Their algorithm is based on the work of Newman and Girvan but improved in terms of computer time and modularity. We used the complete dataset to create the graph where each node consists of a developer or gem. Developers were connected to other developers if they both contributed to the same gem. The above mentioned algorithm was then performed on the graph.

2.4 Survey

To get insights in the qualitative aspects of clusters, we constructed an online survey that was personalized for Ruby developers registered on Rubygems.org. The online survey[1] was able to retrieve information on the respondent's cluster from the database that contained information on developers and gems. For example, if developer a was clustered in cluster k, all other gems and developers in cluster k were shown to developer a when answering the survey.

[1] http://www.ruby-research.com

Data was collected between November 23rd 2012 and February 16th 2013. Invitations to the survey were posted on forums, message boards, and developers were invited to participate by contacting them via email. Email addresses were obtained from public profiles on Rubygems.org.

The variables measured in the survey fall into two categories. The first category measured relationships (ties) towards other developers and other gems residing in the respondent's cluster, and if they had collaborated in producing gems. Furthermore, it measured to what extent they had personal relationships to other developers and, if any, how they were established. The second category measured motivations for creating new gems and the various contributions/roles developers have when creating gems with other developers.

To measure characteristics of relationships (e.g. tie strength), answers could be given on an ordinal scale (e.g. I know no - few - half - almost all - all developers in my cluster). Because our research was explorative of nature, we were not concerned about interval or ratio data to perform statistical analysis. The ordinal scale was sufficient in the sense that it enabled us to make some sort of classification based on the *a priori* grouping of cluster sizes based on the number of developers. Other questions were measured on a nominal scale, such as the various types of relationships or motivations. Moreover, several checkboxes could be selected that were processed with multiple answer analysis.

3 Data

At the time of analysis, the Ruby ecosystem consists of 37,551 gems and 15,679 developers. On average, each gem is created by 2.39 developers. The gems vary in their total downloads from just 48 downloads ("Whitelabel" by Peter Schröder) to a number as high as 12,623,891 downloads ("Rack" by Christian Neukirchen), with a mean value of 16,477.58 and a standard deviation of 238,668.06. Only 113 gems have total downloads exceeding one million, which constitute for 0.3% of the total ecosystem. In contrast, 18,783 gems have equal or less than 1,000 total downloads, representing 50% of the total ecosystem. Out of all the gems, 3.8% had an additional uri to a wiki of their gem, 7.9% provided an uri to additional documentation and only 1.5% provided the option to subscribe to their mailing list.

3.1 Clusters

We have identified 10,586 valid clusters by performing the modularity algorithm proposed by Blondel et al. [11]. The modularity achieved by performing modularity optimization (optimizing the modularity to find distinct clusters) was 0.985, an indication that nearly all clusters are unique, i.e. no developers exist in more than one cluster. These clusters are derived from all 37,551 gems and 15,679 developers. During data analysis, a total of 328 clusters contained a single node, which constituted a gem. These clusters were excluded from our study as, apparently, project administrators failed to register its corresponding

developer(s). A manual search showed that the majority of these clusters are registered as version 0.1, had just a few downloads and provided no additional information. This can be caused by the free nature of its repository, where registering a gem requires no strict rules or formats. An illustration of a random cluster with 7 developers and 13 gems is shown in Fig. 1

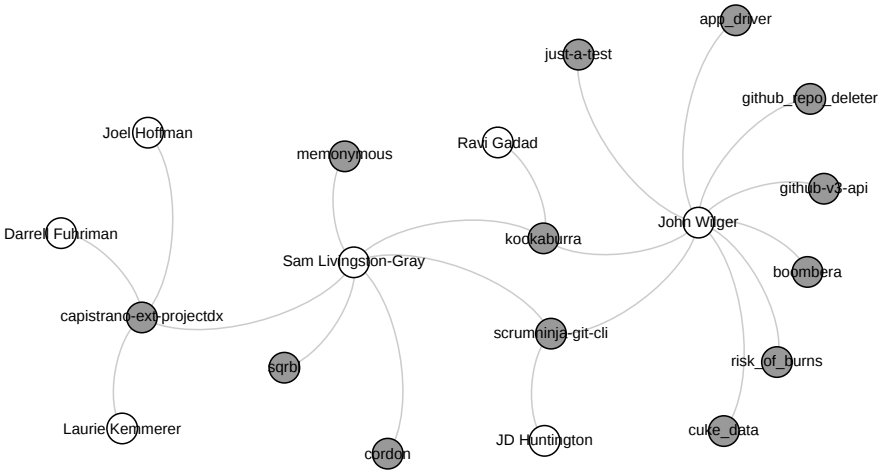


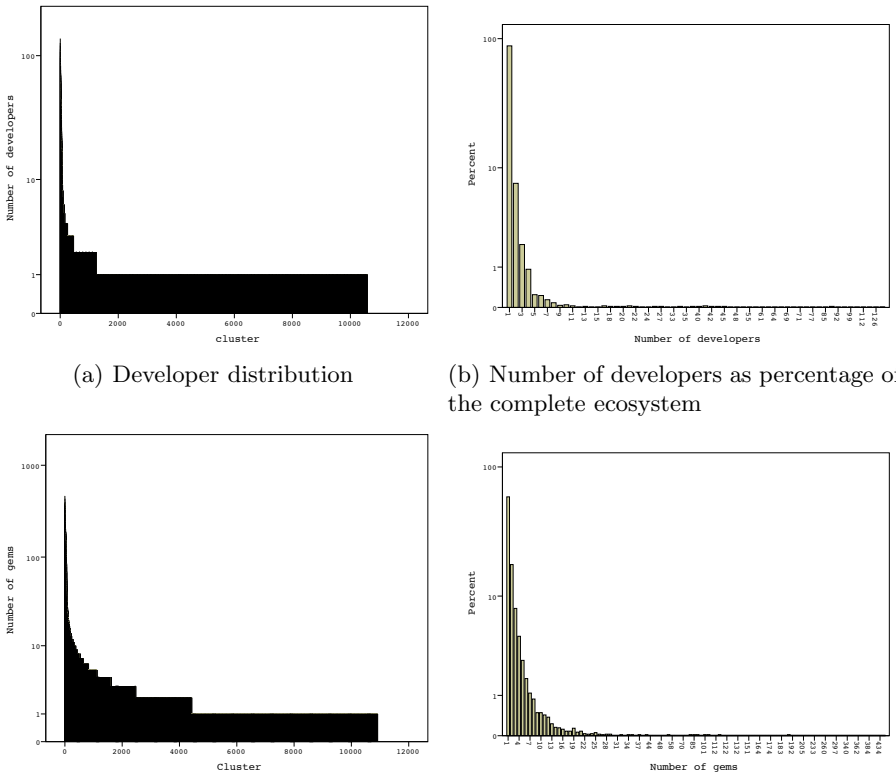
Fig. 1. Cluster with 7 developers (white nodes) and 13 gems (grey nodes) and the relations between them.

Number of Nodes Most clusters encompass just two nodes (5,819 clusters that constitute 55% of the total ecosystem), that is, one developer that is linked to its own gem. A smaller, but still relatively large part, consists of three nodes (2,035 clusters, 19.2%), being a single developer who created two gems, or two developers that have worked on the same gem. The distribution of nodes across clusters shows that a small portion of just 0.5% are clusters with over 100 nodes. In addition, the largest cluster consist of 600 nodes: 136 developers who created 464 gems, which account for less than 0.01% of the total ecosystem. Interestingly, none of Ruby's top 10 most downloaded gems were part of the top 6 largest clusters. "Activesupport", "Activerecord", "Actionpack", "Actionmailer", "Activeresource" and "Rails", created by David Heinemeier Hansson, were part of the 7th largest cluster (107 developers and 370 gems).

Number of Developers/Gems Figures 2(a), 2(b), 2(c), 2(d) show the highly skewed developer and gem distribution across clusters. The skewed distributions further show that the vast majority of clusters consist of just one or two developers. We identified 9,341 clusters with 1 developer (88.2%), 786 clusters with 2

developers (7.4%), and 206 clusters with 3 developers (1.9%). This shows that Ruby's ecosystem, when looking at the number of developers within a cluster, consists of 97.6% of clusters that are not larger than 3 developers. Moreover, similar numbers are found when focusing on the number of gems. Approximately 90% of all clusters have not produced more than 4 gems.

The ample part of the Ruby ecosystem consists of single developers working on one or multiple gems. A smaller, but still relatively large part, consists of two or three developers working together to produce gems. There are few clusters that encompass tens or even +100 developers that are connected through collaboration.



(a) Developer distribution

(b) Number of developers as percentage of the complete ecosystem

(c) Gem distribution

(d) Number of gems as percentage of the complete ecosystem

Fig. 2. Distribution of developers and gems across clusters: y-axis logarithmically transformed

4 Survey

A total of 782 respondents filled in the survey, being approximately 5% of all registered Ruby developers on Rubygems.org. To classify the responses, we grouped clusters by the number of developers within them. As the distribution of developers within a cluster is highly skewed, we created groups with minor increments up to 10 developers. The remainder of clusters, being that with 10+ developers are grouped together. This enabled us to classify survey data on cluster sizes with respect to the number of developers. We excluded clusters of a single developer for questions related to ties, as none were present. The classification of responses are as follows: Clusters with one developer 403 responses, clusters with 2-3 developers 95 responses, clusters with 4-5 developers 35 responses, clusters with 6-7 developers 13 responses, clusters with 8-9 developers 4 responses, and finally, all clusters that have 10 or more developers 232 responses.

4.1 Ties through Collaboration

Tables 1 and 2 show survey data with respect to relationships with other developers that are clustered together. The data of Table 1 provides an overview of ties to other developers. These ties are a mixture of strong and weak ties, for example, developers who have collaborated, have emailed, used each other's gems etc. In addition, Table 2 shows the presence of strong ties, i.e. personal relationships, classified into various cluster sizes.

Two-third of the respondents in clusters with 2-3 developers know all other developers with more than 50% of them having a personal relationship. When looking at large clusters with over 10 developers, the vast majority (84.5%) of developers know just a few other developers, 61.6% of these are personal of nature. Comparing both tables reveals that the relations of strong ties in contrast to ties in general do not drop considerable.

Additionally, we asked, if personal relationships were present, how they were established. The results are shown in Table 3. The large part of these personal relationships are formed through friendships, work mates and through the open source community. In all cluster groupings, they constitute the majority of the responses. Personal relationships that are formed through the open source community seem to grow from 16.5% in clusters with 2-3 developers, to 33.7% in clusters with over 10 developers. In contrast, the percentage of personal relationships that are formed through work associates (work at the same company) seems to diminish from 41% in clusters with 2-3 developers to 25.4% in clusters with 10+ developers. Furthermore, the Ruby ecosystem consists of very few personal relationships that are bounded by family or schoolmates. The respondents that answered "other" were given the option to write out an alternative. More than half of them answered "conference" as a mean to establish personal relationships.

The relationships between developers and gems are based on the author field of the Rubygems.org repository. The algorithm, therefore, seeks for connections

based solely on this information. We asked developers if they would have expected other developers or gems to be clustered within their cluster. From all responses, 36.8% had expected other gems, and 30.8% had expected other developers to be residing in their cluster. The gems that were expected were mostly related to the Ruby on Rails framework, for example, Active Support, Rack, Rails. Additionally, developers seem to have expected other gems on which they have worked through pull requests. Pull requests are a common way of collaborating with others, examples are adding updates or fixing bugs, that are then sent to the project administrators. This does not imply that they list them as an author. Several developers seem to have committed changes to various other projects in which they were not mentioned as an author. Similarly, they would have expected developers that have contributed via pull requests and that were accepted in the codebase.

Table 1. *To what extent do you know the other developers we have identified within your cluster?*

Number of developers	I know a few other developers	I know half of the other developers	I know almost all other developers	I know all other developers	I know only developers with whom I created a gem
2-3	11.6% (n=11)	1.1% (n=1)	2.1% (n=2)	66.3% (n=63)	18.9% (n=18)
4-5	22.9% (n=8)	8.6% (n=3)	14.3% (n=5)	45.7% (n=16)	8.6% (n=3)
6-7	46.2% (n=6)	-	-	30.8% (n=4)	23.1% (n=3)
8-9	25.0% (n=1)	50.0% (n=2)	25.0% (n=1)	-	-
10+	84.5% (n=196)	4.7% (n=11)	.9% (n=2)	-	9.9% (n=23)

Table 2. *To what extent do you personally know the other developers we have identified within your cluster?*

Number of developers	I do not know any of the other developers personally	I personally know a few other developer	I personally know half of the other developers	I personally know almost all other developers	I personally know all other developers	I personally know only developers with whom I created a gem
2-3	18.9% (n=18)	8.4% (n=8)	4.2% (n=4)	2.1% (n=2)	52.6% (n=50)	13.7% (n=13)
4-5	28.6% (n=10)	11.4% (n=4)	11.4% (n=4)	20.0% (n=7)	25.7% (n=9)	2.9% (n=1)
6-7	38,5% (n=5)	7.7% (n=1)	-	-	30.8% (n=4)	23.1% (n=3)
8-9	25.0% (n=1)	50.0% (n=2)	-	25.0% (n=1)	-	-
10+	23.3% (n=54)	6.6% (n=143)	2.2% (n=5)	-	-	12.9% (n=30)

Table 3. *How did you establish your personal relationship (if any) with other developers within your cluster?*

Number of developers:	2-3	4-5	6-7	8-9	10+
Friend	20.9%	16.9%	11.1%	16.7%	19.1%
Work at same company	41.0%	39.0%	22.2%	50.0%	25.4%
Family	2.2%	-	-	-	0.2%
Open Source Community	16.5%	25.4%	27.8%	33.3%	33.7%
Business partner	5.8%	5.1%	11.1%	-	4.8%
Schoolmates	0.7%	5.1%	5.6%	-	2.2%
I work alone / no personal relationship	6.5%	8.5%	5.6%	-	9.2%
Other	6.5%	-	16.7%	-	5.3%

4.2 Motivations to Create New Gems

We asked developers what motivates them when creating new gems (projects). The results are shown in Table 4 with the percentage of responses (adjusted for multiple answers). In each grouping of cluster sizes, the majority of developers are motivated by personal needs, i.e. they need to create a certain gem for their own goals. This does not seem to increase or decrease when clusters consist of more developers. Two other motivations that contribute for a some what smaller part are "helping others" and "improving programming skills". The percentage of responses for "helping others" seems to increase as clusters are getting larger. Furthermore, 'fun' was a frequently given answer in the open text box.

Table 4. *What motivation(s) do you have for creating new gems?*

Number of developers:	1	2-3	4-5	6-7	8-9	10+
Personal needs	65.3%	58.6%	66.7%	48.1%	57.1%	57.9%
To help others	13.5%	14.6%	13.7%	22.2%		18.9%
Monetary rewards	1.8%	2.5%	2.0%	3.7%		3.4%
Improve programming skills	10.8%	11.5%	7.8%	18.5%	14.3%	10.6%
Recognition	6.3%	8.9%	9.8%	7.4%	14.3%	7.0%
Other	2.2%	3.8%	-	-	14.3%	2.3%

4.3 Contribution/role when Collaborating

Lastly, we asked developers about their contribution when creating gems with other developers. Results are shown in Table 5 together with the percentage of cases, as roles vary among different projects. Sixty percent of the cases create gems solely by themselves, an indicator that large parts of the Ruby ecosystem consist of single developers. This was also noticed from the skewed distribution as discussed in Section 3. The remainder of roles are diverse and make up an almost equal part of the cases. Project leaders are, however, in minority. Possibly an indicator that Ruby projects are small in nature and do not consist of a clear hierarchy of developers that are led by a project leader.

Table 5. *What is your contribution when creating gems with other developers?*

Contribution / role	N	Percentage of responses	Percentage of cases
I create gems solely by myself	469	22.40%	60.00%
Testing	275	13.10%	35.20%
Debugging	252	12.00%	32.20%
Documenting	207	9.90%	26.50%
Lending technical knowledge	213	10.20%	27.20%
Project leader	133	6.40%	17.00%
Core developer	239	11.40%	30.60%
Co-developer	282	13.50%	36.10%
Other	24	1.10%	3.10%

5 Discussion and Conclusion

In this paper, we have mined data on relationships between developers and gems from Ruby's repository Rubygems.org. We have used Social Network Analysis to uncover underlying sub-communities, or clusters as we have labeled them. Based on this data, we performed qualitative analysis by means of a survey to make a first attempt to find characteristics of various cluster sizes. These clusters were grouped together by the number of developers residing within them. The survey was aimed at finding characteristics of connections or ties between developers and gems. We analyzed if these connections were personal and how they were established. Furthermore, we made an attempt to classify motivations amongst various cluster sizes and the contributions or roles Ruby developers have when collaborating with other developers.

Clusters with few developers make up a large part of the Ruby Ecosystem. Within these small clusters, developers often have personal relationships that are mainly formed through friendship, work and through the open source community. As clusters are increasing in developer size, the relationships (ties) to other developers, personal and non-personal, seem to decline. It seems that Ruby developers collaborate with other developers, but are not aware of collaborations of friend-of-friend relationships. Additionally, conferences are important to stimulate collaborations between developers, as developers have stated that it facilitated in new collaborations. Personal relationships that are formed through the Open Source community seem to grow as clusters are getting larger. Furthermore, the ample part of the Ruby ecosystem consist of single developers who create gems for personal use and do not engage in interaction.

Our explorative study made a first attempt into the classification of FLOSS data in cluster sizes, but it is however far from complete. As developers have stated, relationships can not solely be modeled by looking at information given by project administrators, as this information lacks detailed information on the actual number of developer contributions. A substantial amount of developers are motivated by altruism and like to help others. As a consequence, developers contribute by making pull requests/issues in which they are necessarily not listed as an author by project administrators. This data is, however, available in other common repositories such as Github. Clustering developers and gems

by incorporating this information would yield more information and a better representation of the actual ecosystem structure.

Another important aspect of the Ruby ecosystem is the existence of dependencies between gems. These dependencies can either be runtime dependencies or development dependencies. The first are other gems that are essential to work in real time for end-users, the latter are gems that are necessary for development purposes [12]. Although developers have not directly collaborated with other developers from gem dependencies, incorporating these dependencies would increase the complexity of the ecosystem and possibly provide a more in-depth view of the actual structure.

The data we collected by mining rubygems.org will be uploaded to Flossmole.org to assist other researchers in the study of FLOSS ecosystems.

6 Acknowledgement

We would like to thank all the Ruby developers who have filled in the survey and especially Jon-Michael Deldin for his comments to this work.

References

1. Jansen, S., Brinkkemper, S., Finkelstein, A.: A sense of community: A research agenda for software ecosystems. In: 31st International Conference on Software Engineering, New and Emerging Research Track. (2009)
2. Messerschmitt, D., Szyperski, C.: Software ecosystem: understanding an indispensable technology and industry. The MIT Press (2005)
3. Xu, J., Christly, S., Madey, G.: Application of social network analysis to the study of open source software. In Bitzer, J., Schroder, P.J.H., eds.: The Economics of Open Source Development. Elsevier (2006)
4. Haythornthwaite, C.: Tie strenghts and the impact of new media. In: Proceedings of the 34th Hawaii International Conference on Systems Science, Maui, Hawaii (2001) 1019
5. Gao, Y., Freeh, V., Madey, G.: Analysis and modeling of the open source software community. In: Proceedings of North American Association for Computational Social and Organizational Science Conference (NAACSOS 2003). (2003)
6. Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M., et al.: Applying social network analysis to the information in cvs repositories. In: International Workshop on Mining Software Repositories. (2004) 101–105
7. Madey, G., Freeh, V., Tynan, R.: The open source software development phenomenon: An analysis based on social network theory. In: Americas conference on Information Systems (AMCIS2002). (2002) 1806–1813
8. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D., eds.: Defining and identifying communities in networks. Number 101, USA, Proc. Natl. Acad. Sci. USA (2004)
9. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69**(2) (2004) 26113
10. Wu, F., Huberman, B.A.: Finding communities in linear time: A physics approach. Eur. Phys. J. **38** (2004) 331–338

11. Blondel, V.D., Guillaume, J., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **10** (October 2008) P10008
12. Kabbedijk, J., Jansen, S.: Steering insight: An exploration of the ruby software ecosystem. In: Proceedings of the Second International Conference on Software Business. (2011)
13. Granovetter, M.: The strength of weak ties. American Journal of Sociology **78**(6) (1973) 1360–1380
14. 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)

Reviewing the Health of Software Ecosystems – A Conceptual Framework Proposal

Konstantinos Manikas and Klaus Marius Hansen

Department of Computer Science (DIKU)
University of Copenhagen
Njalsgade 128
2300 Copenhagen S
Denmark
{kmanikas,klausmh}@diku.dk

Abstract. The health of a software ecosystem is an indication of how well the ecosystem is functioning. The measurement of health can point to issues that need to be addressed in the ecosystem and areas for the ecosystem to improve. However, the software ecosystem field lacks an applicable way to measure and evaluate health. In this work, we review the literature related to the concept of software ecosystem health and the literature that inspired the software ecosystem health literature (a total of 23 papers) and (i) identify that the main source of inspiration is the health of business ecosystems while also influenced by theories from natural ecosystems and open source, (ii) identify two areas where software ecosystems differ from business and natural ecosystems, and (iii) propose a conceptual framework for defining and measuring the health of software ecosystems.

Key words: software ecosystems, ecosystem health, software ecosystem health framework, software ecosystem health measurement

1 Introduction

The notion of software ecosystems (SECOs) is gaining popularity as a means of expanding development, better positioning in the market, or increasing revenues. There is a number of definitions of SECOs in the literature [1, 2, 3, 4]. In this work we define a software ecosystem as *"the interaction of a set of actors on top of a common technological platform that results in a number of software solutions or services. Each actor is motivated by a set of interests or business models and connected to the rest of the actors and the ecosystem as a whole with symbiotic relationships, while, the technological platform is structured in a way that allows the involvement and contribution of the different actors"* [5]. Today, software ecosystems come with a wide variability of characteristics: platform structure, actor participation, ecosystem orchestration, and revenue models, to name a few. This makes the establishment of methods for measuring and evaluating the activity of the ecosystem challenging. The SECO literature refers to the concept of "health" of an ecosystem as a way to monitor ecosystem activity, identify and

predict areas for improvement, and evaluate changes in the ecosystem. However, the measurement of the SECO health is not yet fully achieved.

We tentatively define the health of a software ecosystem as the ability of the ecosystem to endure and remain variable and productive over time. In this work, we aim to get closer to SECO health measurement by reviewing the literature that is elaborating on SECO health. In doing so, we identify that the SECO health literature is borrowing definitions and measurement of health from other fields and expand our literature review focus to include additional ecosystem health fields (explained in section 2). We review the wider ecosystem health literature body and report the health definitions and measurements (section 3). We identify two main differences between SECOs and business and natural ecosystems and, based on previous work, we propose a conceptual framework for defining and measuring the health of SECOs (section 4). Finally, in section 5 we discuss threats to validity and future work and conclude in section 6.

2 Defining the Health Literature Body

The method used for defining the literature body consisted of the following steps:

(i) Defining the SECO health literature. To define the literature related to the SECO health, we used as input the papers identified in our recent systematic literature review [5]. In [5], we identified a number of papers referring to the concept of ecosystem health. The papers have a wide variability on the level of detail they provide on the ecosystem health ranging from mere reference to the concept (e.g., [6, 7, 8]) to papers in which health forms part of the main focus (e.g., [9, 10]).

(ii) Defining wider ecosystem health literature. While examining the SECO health literature, we noticed that the definition and analysis of health is borrowed from other types of ecosystems not covered by the SECO health literature . Using the "snowballing technique" [11], we followed the references of the SECO health literature and evaluated whether these are related to the health of an ecosystem [1]. The criteria for accepting a paper in the literature was that it would (a) define the health or sustainability of an ecosystem or (b) elaborate on ways of measuring health.

Table 1 shows the literature body and the papers that are referenced by each document. The SECO health literature that resulted from step (i) is the first row while the literature from step (ii) are the remaining. We have organized the papers into categories according to their field: software ecosystems (SECOs), business ecosystems (BECOs)[2], natural ecosystems, and open source software (OSS). The main purpose of listing the categories is to show the fields that

[1] We also followed references of the selected references that appeared relevant, resulting in a number of papers ([12, 13])

[2] Paper [27] is defining the field of "IT ecosystems", though, as a BECO with IT products.

Type	Paper	Source
SECO	[14, 10, 15, 9, 16, 17, 7, 18, 8, 19, 20, 21, 6]	Health literature from [5]
BECO	[22]	[14, 9, 21, 10, 15, 23, 24]
	[25]	[14, 7, 10, 16, 6, 23]
	[23]	[14, 10, 9, 16, 18]
	[26]	[18, 8, 23]
	[27]	[20]
	[24]	[9]
Natural ecosystems	[28]	[9]
	[12]	[28]
	[13]	[28]
OSS	[29]	[19]
Total:	23	

Table 1. List of the documents in the health literature and the documents referring to them.

influenced SECO health. In this work, we have not looked at the health definition and measurement in the other fields outside the references of the SECO health literature and, thus, do not claim that these papers are representative of each field.

3 Ecosystem Health

Following the separation of ecosystem fields in Table 1, we list and discuss the papers per ecosystem that have influenced the SECO health literature.

3.1 Natural Ecosystems

The field of (natural) ecosystems inspired the rest of the ecosystem fields examined here (BECO and SECO) and it is the field where the concept of ecosystem health was initially formulated. Costanza [12], defines a healthy ecosystem as "being 'stable and sustainable'; maintaining its organization and autonomy over time and its resilience to stress". In addition, Rapport et al. [28], referring to a collection of papers in the literature, define three indicators for health of an ecosystem: *Vigor* that indicates how active or productive an ecosystem is, *Organization* that indicates the variability of species, and *Resilience* that indicates the ability of the ecosystem to "maintain structure and function in the presence of stress". The characterization of an ecosystem in terms of structure and function is also discussed by Schaeffer et al. in [13]. They parallelize ecosystem health

with human health and define it as the "absence of disease". They identify structure as "numbers of kinds of organisms, biomass etc." and function as "activity, production, decomposition etc.". These are seen as measures used to define the ecosystem health. Furthermore, Schaeffer et al., referring to the literature, list four ways that structure and function may be connected: (a) tightly connected, where neither can change without the change of the other, (b) structure changes does not affect function, (c) function changes do not affect structure, and, (d) structure and function appear unconnected.

3.2 Business Ecosystems

BECO health is the area that has inspired most of the SECO health literature. In the BECO literature, the concept of health is mainly defined as the ability of a BECO to provide "durably growing opportunities for its members and for those who depend on it" [26][3]. Iansiti and Levien [25, 26, 22] and Iansiti and Richards [27] define the health of a business ecosystem using three measures:

Productivity. Inspired by natural ecosystems' ability to create energy from input sources (e.g., sunlight or mineral nutrients), BECO productivity is the ability of an ecosystem to "convert raw materials of innovation into lowered costs and new products and functions" [26]. Productivity in BECOs can be measured by means of (a) total factor productivity, (b) productivity improvement over time, and (c) delivery of innovations, the ability of the ecosystem to adapt and deliver to its members new technologies, ideas, or process.

Robustness. The ability of the ecosystem to sustain shocks, perturbations, and disruptions. Robustness is measured in terms of (a) survival rates, the survival of actors over time, (b) persistence of ecosystem structure, the extent to which actor relationships are kept unchanged, (c) predictability, the extent to which even if shocks alter the relationships of actors, a main core of the ecosystem remains solid, (d) limited obsolescence, whether the ecosystem has a limited invested technology or components that becomes obsolete after a shock, and (e) continuity of use experience and use cases, the extent to which products gradually evolve in response to new technologies rather than changing abruptly.

Niche Creation or Innovation. The ability of the ecosystem to increase meaningful actor diversity over time. Niche creation is measured in terms of (i) growth in company variety and (ii) growth in product and technical variety (value creation) that measures the increase in value the growth brings.

Iansiti and Levien and Iansiti and Richards also propose three ecosystem actor roles, inspired by natural ecosystems, that affect the health of a BECO:

Keystone. Is an actor that normally occupies or creates highly connected hubs of actors and promotes the health of the ecosystem by providing value to

[3] Similar definitions appear in [22, 27]

the surrounding actors. Keystones promote the health of the ecosystem by increasing the variability, provide value to the connected actors and thus increase productivity, and increase robustness by protecting connected actors from external shocks.

Dominators. Are the actors that control the "value capture and value creation" [22] of the ecosystem. They tend to expand by taking over the functions of other actors thus eventually eliminating the actors. Dominators are harmful for the health of an ecosystem as they reduce diversity

Niche (players or firms). Usually form the main volume of the ecosystem actors drawing value from the keystones. A niche player aims to separate from the other niche players by developing special functions.

Typically, a keystone provides value to a number of actors that can be either niche players trying to develop or dominators trying to dominate the functions of the surrounding actors. The roles of the BECO actors are also examined by Iyer and Lee [24]. They classify the actors in an ecosystem in (a) hubs, (b) brokers that connect two sets of actors, and, (c) bridges that are essential for the connectedness of the ecosystem. A hub can demonstrate keystone, dominator, or niche player characteristics.

Hartigh et al. [23] use the work of Iansiti and Levien and Iansiti and Richards (referred to as "Iansiti" hereafter) to measure the health of the Dutch IT industry. They define BECO health using two long-term parameters: the financial well-being and strength of the network and break down the health in two components: partner health and network health. Partner health evaluates the health of each individual actor of the ecosystem. A healthy ecosystem is composed of productive actors contributing to the productivity of the ecosystem while unproductive actors will have difficulty surviving. The survival of the actors is analogous to the Iansiti robustness measure. Network health is measured in terms of actor connectivity. Highly connected actors contribute to the robustness of the ecosystem as the actors are not easily affected by external shocks. In addition, a healthy ecosystem contains clusters of different nature, thus increasing the possibility of niche creation.

3.3 OSS

Wahyudin et al. [29] study the concept of health in OSS projects. They define the health of an OSS project as "survivability", the ability of the project to survive throughout time. An OSS project is healthy and survives if the software produced by the project is used by a number of users and maintained by a number of developers. They identify three measures that affect the health of an open source project:

The developer community liveliness. The project should attract new developers and keep the existing by boosting their motivation. Wahyudin et al. break down an OSS developer's motivation in intellectual stimulation, skill enhancement, and access to source code and user needs.

The user community liveliness. The users of OSS software play an active role in the evolution of the project by reporting bugs and requesting new features. A large, active user community indicates that the software produced is usable and of good quality.

The product quality. A product that is competitive with commercial products in use and quality will attract users and developers, increase the activity in the project, and therefore enhance survivability.

3.4 Software Ecosystems

In the field of SECO, Berk et al. [9] propose SECO-SAM, a model for the assessment of a SECO strategy based on SECO health. In their model, they make an analogy between the health of an ecosystem and human health and propose that SECO health is influenced by the biology of the ecosystem, the lifestyle, the environment, and the intervention of healthcare organizations while they measure the SECO health adopting the Iansiti productivity, robustness, and niche creation (PRN) measures. Jansen et al. [10] elaborate on a three-level model of SECOs, published in [7], consisting of the organization scope level, SECO level, and software supply network level. They define SECO health as a characteristic of the software supply network level using the Iansiti PRN measures. Additionally, they propose the application of the Hartigh et al. [23] measures for defining the health at the SECO level. Angeren et al. [14] show that SECO robustness of the Iansiti PRN measures is an important factor for vendors that choose to depend on a SECO.

In OSS, McGregor [20] translates the Iansiti PRN measures to measures that can be applied to open source projects, while Kilamo et al. [18] propose a framework for going from a proprietary to a Free/Libre/Open Source Software (FLOSS) SECO. One of the framework activities is setting up a "community watchdog" that assesses the community, the software, and "how well the objectives of the company are met". The watchdog indirectly assesses the health while they provide a number of measures to be applied in FLOSS SECOs.

Looking at the SECO health literature, we note that the main source of inspiration is BECO health when trying to define and measure SECO health or health-related parameters (e.g., keystone-dominator strategies) with 11 out of the 13 papers referring to at least one of the Iansiti authored papers [22, 25, 26, 27]. Although the health of a BECO is very similar to the SECO health, we identify a number of differences between the two. In the next section, we explain the differences and build on top of the existing literature to define a framework for SECO health.

4 A SECO Health Proposal

When analyzing the health of SECOs, we identify that similarly to BECOs and natural ecosystems, the set of actors, their activity and the network they form

is an indication of the level of prosperity and sustainability of the ecosystem. However, one main difference of SECOs from BECOs and natural ecosystems is in the nature of the products of the actors and, eventually, of the whole ecosystem. The BECO approach explained in the previous section is aligned with the natural ecosystem approach of actors and products, where the products of the ecosystem (i.e. energy) are represented by the actors (i.e., species) by enclosing energy in the energy flow between the species. In other words, a herbivorous species eats a plant and is eaten by a carnivorous species. This herbivorous species is both an actor and a product in the ecosystem and changes in the health of this species (e.g., number decrease) affects the energy enclosed (product) by this species and, thus, the carnivorous species.

In SECOs, the actors are differentiated from the products. The main product of the actors is software, either as a common software/technological platform, as software components, or services based on software components. The symbiosis of this software can influence the health of a SECO. The influence that the software components have on the SECO health is independent of the actor health. An example would be an actor that creates a software component that enhances the component interoperability and increases the use of the platform and thus contributing to the SECO health. At the same time, this actor might not have a successful revenue model for this software component and end up loosing a big part of the invested effort. The actor will have a negative influence on the SECO health because of low productivity and possibly robustness, while the software component will have a positive influence.

One additional difference of SECOs to BECO/natural ecosystems is that in SECOs there is an entity organizing and managing the ecosystem, the orchestrator. The orchestrator, whether a for-profit organization or an OSS community, is typically managing the ecosystem by running the platform and creating rules and processes for actors and software. The orchestration of the SECO thus has a significant effect on the health of the ecosystem.

The proposed SECO health framework can be seen in Figure 1. We depict three main components that affect the SECO health: (i) the actors, (ii) the software and (iii) the orchestration. In (i), we separate between the individual health of an actor and the health of the network of actors and similarly in (ii) between the individual component health, the ecosystem platform health and the software network health.

4.1 Individual Actor Health

The health of the individual actors influences the overall health of the ecosystem. The actor health can be measured in similar terms to a BECO actor. The actor's productivity and robustness influence the ecosystem. The active participation and engagement of actors brings value to the ecosystem, while the actor's robustness increases the probability that the actor exists and remains involved in the ecosystem activity in the future. If the SECO is a proprietary ecosystem or consists of for-profit organisations, the partner health measures of Hartigh et al. [23] can be directly applied. If in the OSS domain, the actor health can be

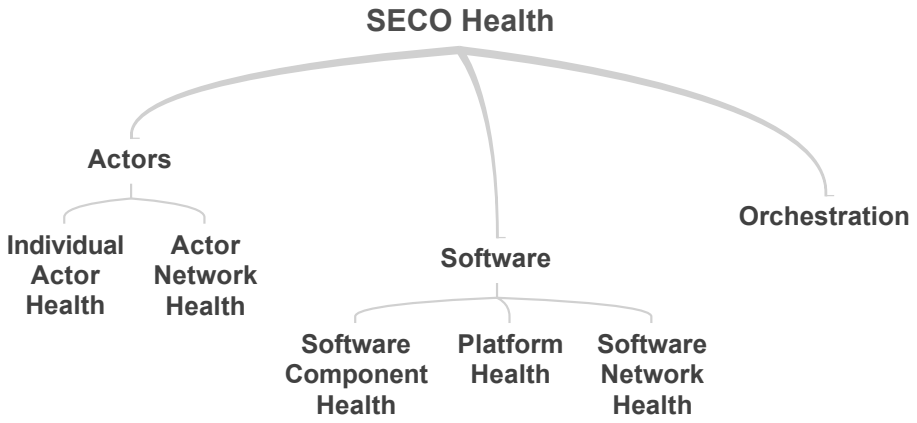


Fig. 1. The SECO health framework breakdown.

assessed in a way similar to Wahyudin et al [29]: measuring the actor activity in the ecosystem (commits, mailing list activity etc.). In that case, an indication of actor robustness is the active participation in the ecosystem over a long period of time. An actor being an active participant in the ecosystem for a long period of time has lower probability of dropping out of the ecosystem than an actor that recently started contributing to the ecosystem.

4.2 Actor Network Health

The network of actors and their interaction plays an important role in the SECO health. The PRN measurements are applicable here, so is the network health perspective of Hartigh et al. [23]. Additionally, the individual actor health may be weighted according to the role of the actor in the network. A keystone with low productivity or robustness will have greater effect in the ecosystem than a niche player with low productivity or robustness.

4.3 Software Component Health

The health of a software component can be measured in terms of, among others, (i) reliability, (ii) availability, (iii) modifiability and prevention of ripple effects, and (iv) interoperability, the ability to interact with, to the extent applicable, the platform and other components. In SECOs, the software components are, in most cases, also the *products* of the ecosystem. The health of such a software component is also influenced by the relative demand and product quality, e.g., how popular is the product and how it is performing in comparison to possible alternatives. This demand is also affected by whether the product is internal, i.e., products intended for use mainly by the ecosystem actors, e.g., the technological platform or external, i.e., products consumed externally to the ecosystem.

4.4 Platform Health

The health characterization of the software components above can be applied to the technological platform of a SECO, since it is a software component itself. However, the technological platform, might have an additional role: depending on how the SECO is organized and managed, the platform reflects possible orchestration actions (rules, processes, or management decisions). The measurement of the platform health should not reflect how the orchestration affects the SECO health (as this is reflected in the orchestration influence on SECO health seen below), but the effectiveness of applying the orchestration actions.

4.5 Software Network Health

The software components are connected and interacting with other components in the ecosystem forming the software network. Graph measures such as connectivity and clustering coefficient show to what extent the components interact [30]. Additionally, the categorization of the activity of hubs into keystone and dominator indicate the level of healthy interaction. Analogous to the Iansiti descriptions in the previous section, an example of keystone activity can be a component that provides interfaces to parts of its functionality for the neighboring components to consume, while in a dominator activity the component would intent to take over functionality of the neighboring components.

4.6 Orchestration Influence to Health

The orchestrator can monitor the health of the ecosystem and take measures to promote ecosystem health if necessary. This requires that the orchestrator has a good overview of the ecosystem and is consulting effective measurements (e.g., ecosystem health). Additionally, the orchestrator can act by creating/refining rules and processes for the actors, communicating plans to the actors (e.g., by road-mapping), organizing the ecosystem development through, e.g., release management, making changes to the platform and other software components, changing the revenue model for internal products, and controlling the actor population and motivation by modifying the model by which the actors participate in the ecosystem. The orchestration of a SECO, i.e., the actions of the orchestrator, possibly based on monitoring and evaluation, influences SECO health.

4.7 Other Influences on SECO Health

Additionally, there might also be influences on the SECO health that are external to the ecosystem. This kind of influences are referred to as "(external) perturbation" in the literature [28, 22, 26, 27] and are disturbances that are outside the control of the ecosystem actors. Influences of this kind might be the establishment or rise of a competitive ecosystem or a radical technological or legal change.

5 Threats to Validity and Future Work

The wider ecosystem health literature used in this study was identified through the references in the SECO health literature as our focus was literature that influenced the SECO health literature. As already mentioned, the literature on each field (apart from SECO) is not necessarily the representative or most influential work in the field. Identification of the most influential work and possible literature mapping of the health in each of the fields (BECO, natural ecosystem, OSS/FLOSS) might bring perspectives into the SECO health that have been overlooked. Additionally, we speculate that the influence of the difference fields to the SECO health is not necessary reflected in the number of papers appearing in this work. Natural ecosystem have had a greater impact on SECO health concepts, but most of it is indirect through the health of BECOs.

Moreover, the proposed conceptual framework, at this point, does not go into detail on the different kinds of actors. An expansion of the model would further analyze on the nature of the actors, e.g., developing companies, resellers, value-adding-resellers, and possibly include their influence on health. Additionally, although the model included products, it did not include customers or end-users. The influence of entities of this kind could be discussed in future work.

6 Conclusion

In this paper, we analyzed the concept of software ecosystem (SECO) health. In order to define SECO health and its measurement, we examined the SECO health literature, a literature body of 13 papers touching upon the concept of SECO health. We identified that the health research is mainly inspired by three fields: business ecosystems (BECO), natural ecosystems, and open source software, with BECO being the main source of inspiration in 11 out of the 13 SECO health papers. We reviewed the wider ecosystem health literature, consisting of 23 papers, explained how they define and measure the health of an ecosystem and concluded with two contributions: (i) We identify two differences between the SECO and business and natural ecosystems: (a) they perceive products in the ecosystem differently. BECOs and natural ecosystems perceive actors as a product per se, while in SECOs an actor produces software components or services. (b) SECOs have an orchestrator entity managing the ecosystem, something that does not appear in the BECO/natural ecosystem literature. (ii) We propose a logical framework for defining and measuring the SECO health consisting of the health of (a) each individual actor, (b) network of actors, (c) each individual software component, (d) platform, (e) software network, and (f) orchestrator. The purpose of this study is to create a discussion on the particularities of SECO health and bring the community closer to a measurable way of defining the health of software ecosystems.

Acknowledgements

This work has been partially funded by the Connect2Care project[4].

References

1. Jansen, S., Finkelstein, A., Brinkkemper, S.: A sense of community: A research agenda for software ecosystems. In: Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on. (may 2009) 187 –190
2. Bosch, J., Bosch-Sijtsema, P.: From integration to composition: On the impact of software product lines, global development and ecosystems. Journal of Systems and Software **83**(1) (2010) 67 – 76
3. Messerschmitt, D., Szyperski, C.: Software ecosystem: understanding an indispensable technology and industry. MIT Press Books **1** (2003)
4. Lungu, M., Lanza, M., Gîrba, T., Robbes, R.: The small project observatory: Visualizing software ecosystems. Science of Computer Programming **75**(4) (2010) 264 – 275 Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008).
5. Manikas, K., Hansen, K.M.: Software ecosystems – A systematic literature review. Journal of Systems and Software **86**(5) (2013) 1294 – 1306
6. Mizushima, K., Ikawa, Y.: A structure of co-creation in an open source software ecosystem: A case study of the eclipse community. In: Technology Management in the Energy Smart World (PICMET), 2011 Proceedings of PICMET '11:. (august 2011) 1 –8
7. Boucharas, V., Jansen, S., Brinkkemper, S.: Formalizing software ecosystem modeling. In: Proceedings of the 1st international workshop on Open component ecosystems. IWOCE '09, New York, NY, USA, ACM (2009) 41–50
8. Viljainen, M., Kauppinen, M.: Software ecosystems: A set of management practices for platform integrators in the telecom industry. In Regnell, B., Weerd, I., Troyer, O., Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C., eds.: Software Business. Volume 80 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2011) 32–43 10.1007/978-3-642-21544-5_4.
9. van den Berk, I., Jansen, S., Luinenburg, L.: Software ecosystems: a software ecosystem strategy assessment model. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ECSA '10, New York, NY, USA, ACM (2010) 127–134
10. Jansen, S., Brinkkemper, S., Finkelstein, A.: Business network management as a survival strategy: A tale of two software ecosystems. In: First International Workshop on Software Ecosystems (IWSECO-2009), Citeseer (2009) 34–48
11. Denscombe, M.: The good research guide. Open University Press (2010)
12. Costanza, R.: Toward an operational definition of ecosystem health. Ecosystem health: New goals for environmental management (1992) 239–256
13. Schaeffer, D.J., Herricks, E.E., Kerster, H.W.: Ecosystem health: I. measuring ecosystem health. Environmental Management **12**(4) (1988) 445–455

[4] `http://www.partnerskabetunik.dk/projekter/connect2care.aspx`

14. van Angeren, J., Blijleven, V., Jansen, S.: Relationship intimacy in software ecosystems: a survey of the dutch software industry. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems. MEDES '11, New York, NY, USA, ACM (2011) 68–75
15. dos Santos, R.P., Werner, C.M.L.: A proposal for software ecosystem engineering. In: Third International Workshop on Software Ecosystems (IWSECO-2011), CEUR-WS (2011) 40–51
16. Jansen, S., Brinkkemper, S., Souer, J., Luinenburg, L.: Shades of gray: Opening up a software producing organization with the open software enterprise model. Journal of Systems and Software **85**(7) (2012) 1495 – 1510
17. dos Santos, R.P., Werner, C.: Treating business dimension in software ecosystems. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems. MEDES '11, New York, NY, USA, ACM (2011) 197–201
18. Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T.: From proprietary to open source-"growing an open source ecosystem". Journal of Systems and Software **85**(7) (2012) 1467 – 1478
19. Dhungana, D., Groher, I., Schludermann, E., Biffl, S.: Software ecosystems vs. natural ecosystems: learning from the ingenious mind of nature. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ECSA '10, New York, NY, USA, ACM (2010) 96–102
20. McGregor, J.D.: A method for analyzing software product line ecosystems. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. ECSA '10, New York, NY, USA, ACM (2010) 73–80
21. van Ingen, K., van Ommen, J., Jansen, S.: Improving activity in communities of practice through software release management. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems. MEDES '11, New York, NY, USA, ACM (2011) 94–98
22. Iansiti, M., Levien, R.: The keystone advantage: what the new dynamics of business ecosystems mean for strategy, innovation, and sustainability. Harvard Business Press (2004)
23. den Hartigh, E., Tol, M., Visscher, W.: The health measurement of a business ecosystem. In: Proceedings of the European Network on Chaos and Complexity Research and Management Practice Meeting. (2006)
24. Iyer, B., Lee, C.H., Venkatraman, N.: Managing in a small world ecosystem: Some lessons from the software sector. California Management Review **48**(3) (2006) 28–47
25. Iansiti, M., Levien, R.: Strategy as ecology. Harvard Business Review **82**(3) (2004) 68–81
26. Iansiti, M., Levien, R.: Keystones and dominators: Framing operating and technology strategy in a business ecosystem. Harvard Business School, Boston (2004)
27. Iansiti, M., Richards, G.L.: The information technology ecosystem: Structure, health, and performance. Antitrust Bull. **51** (2006) 77
28. Rapport, D., Costanza, R., McMichael, A.: Assessing ecosystem health. Trends in Ecology & Evolution **13**(10) (1998) 397–402
29. Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., Tjoa, A.M.: Monitoring the health status of open source web-engineering projects. International Journal of Web Information Systems **3**(1/2) (2007) 116–139
30. Hansen, K.M., Manikas, K.: Towards a Network Ecology of Software Ecosystems: an Analysis of two OSGi Ecosystems. In: Proceedings of the 25th International Conference on Software Engineering & Knowledge Engineering (SEKE'2013). (2013)

On the Software Ecosystem Health of Open Source Content Management Systems

Sonny van Lingen[1], Adrien Palomba[1], Garm Lucassen[1]

[1]Utrecht University
{s.j.vanlingen, a.r.v.palomba, g.g.lucassen}@students.uu.nl

Abstract. Choosing a content management system on which you rely your business is challenging because they need a healthy software ecosystem in order to function efficaciously. Unawareness of this will result in content managers having uncertainty about the future suitability of their chosen content management system. This study describes an empirical, inductive approach by comparing the software ecosystem health of the three most popular open source Content Management System platforms (WordPress, Joomla and Drupal) according to a number of health characteristics. Taking the software ecosystem health of a desired content management system into account enables stakeholders to make a more grounded decision in choosing either of the Content Management Systems. This could lead to a more suitable, dynamic and/or sustainable solution.

Keywords: software ecosystems, software ecosystem health, Drupal, Joomla, WordPress, content management systems

1 Introduction

Using an online Content Management System (CMS) to create and add content to a dynamic website is increasingly growing popular [14]. Designing an attractive website with the help of an online CMS is done with much more ease than having to perform a manual hard-coding process. A large amount of CMS platforms are offering turnkey solutions; however, specific features are mostly not available in a basis CMS installation package. In this case the content manager (website administrator) resorts to additional plugins. Plugins are collections of files developed by a third party, adding functionality to the core of the CMS platform. Therefore, the CMS platforms and the responsible developers for writing third party modules are part of a software ecosystem. Software ecosystems are defined by Jansen [6] 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.' Not being able to survive in a software ecosystem has already led to the demise of many software vendors [6]. Being a CMS platform, measuring the health of your own software ecosystem is essential. More so, for content managers who have to decide on implementing either one of the CMS platforms, this work can help in

making a sensible choice for either one of the CMS platforms (as the software ecosystem's health characteristics relate to its lifetime expectations). WordPress, Drupal and Joomla all act as software ecosystem orchestrators (in this context, as a vendor) by providing third party module developers the opportunity to develop plugins within an open platform. This has lead us to pursue the following research question:

What is the health of the Software Ecosystems of the three most popular open source content management systems?

It is necessary to understand that WordPress is responsible for a significantly higher market share *(53,6%)* than both Joomla *(9,6%)* and Drupal *(6,4%)* [18] and that WordPress' community of third party developers is not comparable to both the Drupal and Joomla ecosystem in terms of development maturity. It demonstrates that Drupal and Joomla are in a battle for the second spot in the open source CMS market, behind WordPress. Furthermore, it is necessary to understand that we consider content managers who are not involved in developing modules not to be active contributors in the ecosystem because they do not make an active contribution - they are solely using the system.

The practical contribution of this research is to provide detailed information that describes the software ecosystem health of the three CMS platforms at both a platform-level and a module-level. This is done by measuring a number of software ecosystem health characteristics, which are described elaborately in the research method section. This is done by a mixture of computational calculations, our own observations and a survey (to confirm the aforementioned findings). In gathering data and response for the survey we heavily relied on communities (forums) and plugin overviews on the official sites of the three platforms' websites. Communities and plugin overviews of unofficial, third party sites are not taken into account in this research as their reliability (and their completeness) could be questioned.

This section introduced the notion of software ecosystems, software ecosystem health and its relation to CMS platforms. The remainder of the paper is structured as follows: In Section 2 a literature study is described, which defines key terms and provides definitions, besides identifying studies that support our research topic. Section 3 details the research methodology that was applied. Section 4 presents an analysis of the results together with the findings of our research. Finally, the conclusions, the limitations and an outlook for further research are provided in Sections 5 and 6.

2 Related Literature

In order to comprehend and explain the context of our study we carried out a focused literature review. In this section we consecutively describe three different aspects that have an important relationship with the software ecosystem health of CMS platforms, namely: software ecosystems, software ecosystem health and finally CMS platforms (and comparisons thereof).

It is observed that the notion of software ecosystems is still remarkably young; the first definitions are coined in the year 2003. However, up until 2008 the concept of ecosystems in a software or information technology perspective was still considered "not directly obvious" [11]. Various definitions of the notion of software ecosystems exist [2, 6, 10, 11]. We however consider the definition of Jansen the most appropriate to this study, which can be found in detail at the introduction section of this work.

According to Jansen, Brinkkemper and Finkelstein software ecosystems can be one of the following types: (1) market, (2) technology, (3) platform and (4) firm [7]. Within each type there are a number of factors that can help in reducing the scope of the software ecosystem. This study can be placed in the third category; this study's goal is to compare the software ecosystem health of three CMS platforms. Jansen and Cusumano [8] provide a classification model for software ecosystems, which is applied to 19 cases previously explored in software ecosystem literature. Finally, Campbell and Ahmed propose an elaborated three-dimensional view on the software ecosystem model explained by three central pillars: business; architecture and social aspects [3]. Software ecosystem health indicators are part of a software ecosystem related survey carried out under representatives of the Dutch Software industry [1].

As early as 2003, McKeever recognized the shift from static, manually deployed web content to dynamic, automatically deployed web content and the potential of content management systems in this perspective [12]. The maturity of CMS's has grown due to new web technologies, plus the need for improved role based web management that has supported this growth [16]. This growth in maturity and popularity has resulted in the fact that ~31.7% of today's websites are managed by a CMS platform [17]. Some works already compared CMS platforms by using other, non-ecosystem-related metrics. In a Search Engine Optimization (SEO) comparison experiment of the Joomla, Drupal and Wordpress CMS platforms, Drupal came out as the platform generating the most search engine revenue (2099 unique visitors from search engines in six months), followed by Joomla (1619 visitors) and WordPress (1439 visitors) [15].

A performance analysis of CMS platforms, again comparing Joomla, Drupal and Wordpress, reveals that the Joomla platform is best suited for novice content managers, whereas Drupal is suited better for content managers having to perform critical tasks and having to provide an increased flexibility [9]. A security audit report detailing the technical security of the Joomla and Drupal platform revealed unpleasant results; as of August 2009 the platforms were considerably safe but both platforms possessed a number of threatening security malfunctions [13]. Although it is not formally confirmed by another research engagement that these security malfunctions are not to be seen anymore, it is more than likely that these security threats are fixed at this moment in time.

3 Research Method

Reviewing the software ecosystem health of the three CMS platforms has led us to decide on a number of software ecosystem health metrics, partially inspired by economic ecosystem health characteristics [5] namely: (1) niche creation, (2) productivity and (3) robustness. A number of these health metrics are computationally measured,

which puts us in the position to process large amounts of data which would otherwise be impossible to review. Furthermore, a number of health metrics are measured manually. Finally, to confirm our findings, we carried out a brief survey under members (website administrators, module developers, core developers) of the ecosystems of interest, researching how they perceive the ecosystem health of the platform of choice. To this end we retrieved a random sample of respondents of interest. This sample consists of members of the three platform's community forums, workshop participants[1] and the authors' professional relations. The complete list of health metrics looks as follows:

- Growth of the platform (computational)
- Identification of the contributors (computational)
    - Including the number of unique developers.
- Up-to-dateness of modules (computational)
- Findability of the ecosystem (computational)
- Centrality of the platform (manual)
- Market share analysis (manual)
- Level of contribution per community user (manual)
- Perceived ecosystem health (survey, manual)

In order to perform the computations needed for the computationally measured health metrics we have developed a set of tools using either the PHP or Java platform. All of these tools exploit the mechanism of HTML parsing, which consists of browsing the HTML code of a given page to seek for a given value, since neither WordPress's, Joomla's nor Drupal's platform offer an Application Programming Interface (API) for executing search queries.

All of these programs have been executed from servers within the Netherlands, all using exclusively Dutch IP addresses[2]. During one encounter we faced a call limit per IP address. This has been solved by resorting to a VPN service which allows changing the external IP address on set intervals. A pool of exclusively Dutch IP addresses has been used for this purpose. The data gathering process started on 28 December 2012 and ended on 3 January 2013. Data originating from the year 2013 is filtered as we are only taking entries up to 31 December 2012 into account during the analysis. This has been decided to assure the analysis has a consistent end-date for all three platforms. We retrieved two collections of data:

- All official extensions for WordPress, Drupal and Joomla including every relevant field provided on its originating website (including name, author, date of creation and date of last modification).
- The number of Google hits per individual module.

The data utilized for measuring the manually measurable health metrics did not include computational interference - this data was accessible in a usable format right away.

[1] Participants in the 'Dutch Student Workshop on Software Ecosystems 2013'.

[2] Hereby avoiding retrieving different results given different geographical ranges of IP ranges.

4 Results and Data Analysis

This section presents our findings and the data analysis. These results are provided in table 1. Finally, to confirm our findings, we provide the results of the survey. These results are provided in table 5.

Table 1. Results overview per ecosystem health metric.

HEALTH METRIC	RESULTS
Growth of the platform in modules	**Drupal** has (and always has had) a larger number of modules for within its platform. Both **Joomla** and **Drupal** have shown a rapid growth after their respective introduction years. *Additionally visualized in figure 1.*
Growth of the platform in number of developers	The **Drupal** platform had always had a larger number of unique developers than **Joomla**, except for a limited period in 2010. During this period, **Drupal** failed to provide core updates for two years. *Additionally visualized in figure 2.*
Identification of the contributors	Total number of developers as of January 2013: **WordPress** *(9,904)* **Drupal** *(6,309)* **Joomla** *(3,360)*
	Average number of modules per developer as of January 2013: **Drupal** *(3.09)* **Joomla** *(3.01)* **WordPress** *(2.31)*
Up-to-dateness of modules	Percentage of modules updated in the year 2012: **Drupal** (59.62%) **Joomla** (41.57%) **Drupal including sandbox** (41.53%) **WordPress** (44.62%)
Findability of the ecosystem	**Joomla's** findability decreases from the year 2010 and on. **WordPress's** findability increases from that point. **Drupal** remains a smaller, niche player. *Additionally visualized in figure 3.* **Drupal** and **WordPress** show only a few cases of unfindable modules (~5%). **Joomla** suffers of approximately half its modules not generating results. **WordPress** seems to operationalize a slightly more effective SEO strategy. *Additionally elaborated upon in table 2.*
Centrality of the platform	**Drupal** is the most centralized platform, followed by **WordPress** and **Joomla**. *Additionally elaborated upon in table 3.*
Market share	**WordPress** possesses the largest market share (53,6%), **Joomla** (9.6%) and **Drupal** (6,4%) are battling for the second spot [18].
Level of contribution	**WordPress's** forum community possesses the largest number of topics and posts. *Additionally elaborated upon in table 4.*

A couple of remarks are to be made considering these results. Sandbox modules are modules which are not fully operational (yet). In the first two health metrics, WordPress could not be included as the platform does not publish the module's date of creation. The 2010 deviation, as can be seen in figure 2, could be explained by an ecosystem transfer of unsatisfied Drupal developers migrating to the Joomla ecosystem. During this period Drupal failed to provide core updates for two years whereas Joomla was releasing a major beta.

Currently, WordPress attracts more developers to join their ecosystem. On average however, the WordPress module developers are slightly less productive. In analyzing the up-to-dateness of modules, the effect of including sandbox modules in this analysis is remarkably. Considering sandbox modules, the up-to-dateness is equal over all three platforms. Furthermore, in analyzing the findability of the ecosystems, we made use of Google's Trends functionality to gain an insight in the platform's findability. In analyzing the findability of individual modules we resorted to a self-developed tool, performing Google searches based on module name. Because there is no consensus on the term module (plugin and extension are also often used), we included all three terms. Scores have been normalized in order to remove false positives (only including scores where $-3 < z\text{-}score > 3$). The query used has the following form:

```
"module MODULE_NAME" OR "plugin MODULE_NAME" OR "extension
MODULE_NAME" + CMS_NAME"
```

In analyzing the centrality of the platforms, we based our findings on the platform's official communication channels. In analyzing the market shares of the platforms, an important remark has to be made: the number of weekly downloads of the platform's executable is declining vastly (WordPress -34,4%, Joomla -24,0%, Drupal -32,2%) [18]. This might imply that the open source CMS market has already matured. Finally, in analyzing the level of contribution, WordPress could not be included entirely. Their forum community does not publish detailed information per member.

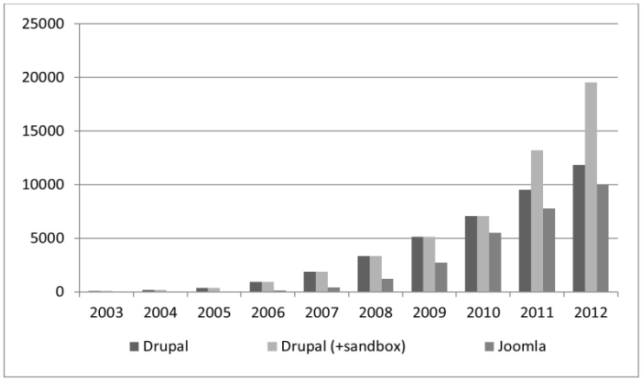


Fig. 1. Growth of the number of modules per CMS platform.

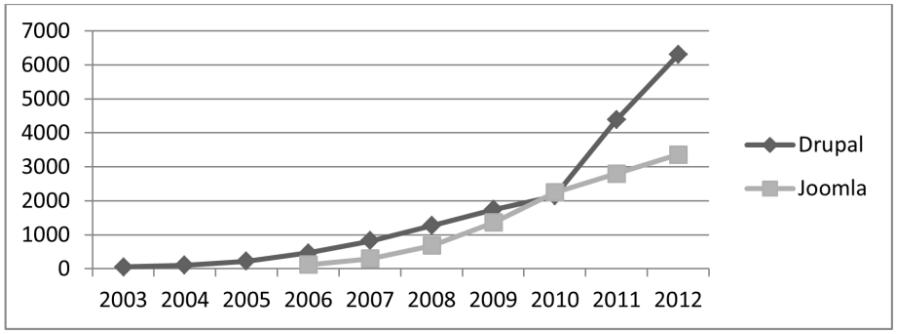


Fig. 2. Growth of the number of unique module developers per CMS platform.

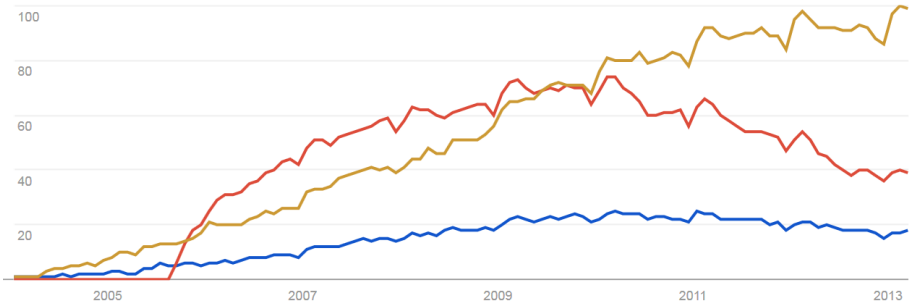


Fig. 3. Google Trends analysis of findability of the platform [4]. *From the bottom and upwards (at the commencing of 2013): Drupal, Joomla, WordPress.*

Table 2. Descriptive statistics of module findability on Google.

	Drupal	***Joomla***	***WordPress***
AVG	5,447	4,856	6,118
MAX	1,720,000	1,640,000	1,950,000
MIN	0	0	0
STD DEV	48,235	56,996	51,098

Table 3. Centrality of the platforms.

	Drupal	***Joomla***	***WordPress***
Module centrality	**Y**	N	**Y**
Support forum centrality	**Y**	**Y**	**Y**
Organized event centrality	**Y**	**Y**	**Y**
Documentation centrality	**Y**	**Y**	**Y**
Availability of support (platform-level)	**Y**	N	N
Availability of support (module-level)	**Y**	N	**Y**

Table 4. Descriptive statistics of the platform's forum communities.

	Drupal	***Joomla***	***WordPress***
Total number of members	1,334,960	593,517	*irretrievable*
Total number of topics	301,098	647,853	913,912
Total number of posts	1,115,507	2,718,144	3,090,335
Average number of posts per member	0.8357	4.580	*incalcuble*
Average number of topics per member	0.2256	1.094	*incalcuble*

4.1 Perceived Ecosystem Health

In the previous subsections we described and measured a number of ecosystem health metrics. The outcome of these measurements serves as a factual, raw data dependent mean to measure the given ecosystems' health. In order to compare these findings to how a number of stakeholders *(n=23)* perceive the ecosystem's health, we carried out a brief survey. Beforehand, it is hypothesized that a substantial amount of respondents regard the ecosystem health of their platform of choice equivalently to the findings presented previously. Furthermore, it is hypothesized that respondents also identify themselves with the poor prospects of the 'traditional' notion/operationalization of CMS platform providers and the upcoming shift to SaaS CMS solutions. The survey's outcome is described in table 5. Note that some comments were rephrased because they were either in Dutch or otherwise in a format not fit for citing. We tried to apply this rephrasing as sharp and precise as possible. We have removed entries from respondents who did not complete the survey in a professional, plausible way.

A couple of noteworthy comments were given. One respondent is unsure of our chosen cloud naming convention, stating that cloud based CMS solutions should be referred to as PaaS. We however do not second this vision – such cloud based solutions do seldom offer an actual platform. Furthermore, one respondent is using WordPress and Joomla and considers WordPress to be much easier for end-users than Joomla. Finally, one respondent is reluctant to embrace SaaS CMS solutions. Additional cost is not seen as the major drawback:

"I always want to host my website and data myself. I am absolutely not comfortable with SaaS suppliers being able to access my (personal) websites and data. "

A couple of remarks are to be made considering these results. In the first question, the distribution of the results is similar to the platform's respective market shares. In the second question, respondents could select more than one checkbox so these numbers, in total, exceeds the number of respondents.

One respondent is already actively migrating websites to other platforms, whereas another respondent feels that WordPress needs to revise their strategic decision about the (lack of) templates.

Table 5. Summarized representation of the survey's outcome (r = number of respondents).

QUESTION	RESULTS
1. Which CMS solution are you currently most involved with?	WordPress (r=10) Joomla (r=7) Drupal (r=6)
2. In which way are you currently involved in the previously selected CMS platform?	Content Manager (r=22) Module developer (r=6) Platform core developer (r=3)
3. Are you worried about the (future) well-being of the previously selected CMS platform?	Not at all, complete trust (r=15) I have reasonable doubt (r=7) I will drop the platform as soon as possible (r=1; Drupal)
4. Have you heard about cloud computing SaaS (Software as a Service) CMS solutions?	Yes, I have (r=11) No, I have not (r=12)
5. Are you currently planning on migrating to another CMS solution?	Yes, I am (r=3) No, I am not (r=20)
6. Most SaaS CMS solutions are paid services. Assuming they suit your demands better, would you consider migrating to them despite the additional cost?	No, I will not consider paying (r=10) Yes, paying for a services does not bother me (r=5) Maybe, I will first need to have more information (r=8)

When asked for SaaS CMS solutions already known to the respondents, a number of solutions were named explicitly:

- WordPress (4 times);
- Netfirms.com;
- Shopify;
- Silkapp;
- Square Space;
- Google Sites;
- TransIP;
- LightCMS;
- WIX.

Three respondents declared they are currently planning on migrating to another CMS solution. When asked for clarification, the first mover declared to be migrating to the Drupal platform. The second mover declared to be moving to an unnamed SaaS CMS environment. The third and last mover declared to be moving to a self-developed CMS.

Finally, a substantial amount of respondents does not feel informed sufficiently about the potential of SaaS CMS solutions (*r=8* out of a sample of *n=23*).

5 Discussion

For this research to become more mature and to allow for a more powerful comparison of the platforms, future research could be devoted to retrieve and analyze more historical data about the platforms and its modules. Even though the data set used for this research was large and detailed, we encountered some limitations within this research. Therefore completeness is not claimed.

Firstly, we could only resort to publicly available data. In spite of the fact that this allowed for a rich 'snapshot' of the software ecosystem's health during the period in which the data was gathered, we had limited access to historical data. In the context of this research, more historical data would have proven to be useful.

Secondly, a number of other software ecosystem health characteristics are not elaborated upon. This is, on one hand, related to restrictions of the data available. The most apparent deficiencies of the data are the lack of a comparable number of downloads per module for the platforms and the lacking possibility to download (and analyze) modules computationally (thus, automatically). Due to the fact that Joomla decentralized the hosting of modules we were unable to retrieve these modules computationally, disqualifying them for automated code analysis. On the other hand, the chosen health characteristics, metrics and its subsets are based on the authors' intuition and expertise. These characteristics do not necessarily follow theoretical classifications and considerations in the soundest way, which might have resulted in missed opportunities in the selection and/or operationalization of health characteristics.

Thirdly, it is to say that the number of Google hits representing a particular module could be questionable, as it might have triggered an unknown number of false positive results. Normalization of the results still does not guarantee that we succeeded in excluding all false positives. However, this eliminated a large part of the outliers.

Finally a remark is to be made about the platform's end-user base. A large number of WordPress' SaaS-users are using the platform as a (personal) blogging tool – opposed to a relatively larger number of professional appliances by their competitors. Due to feasibility reasons these differences in 'content-maturity' have not been analyzed.

6 Conclusion

The main goal of this paper was to measure and compare the software ecosystem health of the Drupal, Joomla and WordPress CMS platforms. This has been done by empirically measuring a number of health metrics, for which we computationally and manually retrieved data. The focus of this comparison was at a platform-level and a module-level.

The results show that the Joomla and Drupal platforms have a comparable market share. Both market shares are significantly smaller than that of the market leader, WordPress. Furthermore, the results show that Drupal's level of growth has exceeded

Joomla's level of growth. Unfortunately, we were unable to retrieve comparable data for the WordPress platform, which would have enabled us to elaborate on the growth of this platform's number of modules and unique developers. Next to this, it is observed that the full-project modules within Drupal's platform are more up-to-date than Joomla's and WordPress' modules (that is, excluding Drupal's sandbox modules). Including these sandbox modules makes the three platforms' up-to-dateness surprisingly equal. Finally, it is observed that Drupal's platform is more centralized than Joomla's and WordPress platform.

Despite the fact that not all metrics are in favor of Drupal's platform, we conclude that Drupal's platform possesses a healthier ecosystem. Hereby it is taken into account that the results for the WordPress platform could not be properly supplemented to two of the health characteristics. These results lead us to conclude that the criteria used by the CMS users to choose a CMS are not primarily based on the health of its ecosystem. Furthermore, given our investigation on Google hits for modules and the platform as a whole, neither of these criteria seem to be a nontrivial criterion for users.

That said a few remarks are to be made. Firstly, the most recent Google's Trends analysis shows a slight downward trend for Drupal's and Joomla's CMS platforms. This suggests that both platforms already have matured and might lose a factor of their popularity in coming years. Furthermore, it is to be noted that the average number of weekly downloads declined vastly for these platforms. This implies that both ecosystems are in the process of becoming unhealthier, or that the open source CMS market is experiencing a (temporary) loss of popularity. However, this does not affect the WordPress platform.

To summarize: based on this research, Drupal's platform is the healthier one of the three platforms, despite of being the least popular. The results of the survey give to think that SaaS CMS solutions have not yet become a threat to "classical" CMS's. Solutions of this kind will probably mature in the future and will require new investigations to quantify its evolution.

References

1. Van Angeren, J., Blijleven, V. and Jansen, S.: Relationship Intimacy in Software Ecosystems: A Survey of the Dutch Software Industry. Proceedings of the Conference on Management of Emergent Digital Ecosystems (MEDES 2011) .
2. J. Bosch. From software product lines to software ecosystems. In Proceedings of the 13th International Conference on Software Product Lines (SPLC) . Springer LNCS, 2009.
3. Campbell, P.R.J., Ahmed, F.: A Three-Dimensional View of Software Ecosystems. Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (ECSA '10). 81–84 (2010).
4. Google: Google Trends, http://www.google.nl/trends/explore#q=drupal, joomla,wordpress, (2013).

5. Iansiti, M., Levien, R.: The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability. Harvard Business Review. 20, 2, 1 (2004).
6. S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. 31st International Conference on Software Engineering, New and Emerging Research Track , pages 187–190, 2009.
7. Jansen, S., Finkelstein, A., and Brinkkemper, S. Business network management as a survival strategy: A tale of two software ecosystems. In Proceedings of the First Workshop on Software Ecosystems. CEUR-WS, vol. 505, (2009)
8. Jansen, S., Cusumano, M.: Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance. Proceedings of the international Workshop on Software Ecosystems 2012. 1–18 (2012).
9. K Patel Savan Rathod V R Prajapati, J.B.: Performance Analysis of Content Management Systems- Joomla, Drupal and WordPress. International Journal of Computer Applications 0975 – 8887. 21, No.4, (2011).
10. Kittlaus, H.-B., Clough, P.N.: Software Product Management and Pricing: Key Success Factors for Software Organizations. Springer (2009).
11. Kuehnel, A.-K.: Microsoft, Open Source and the software ecosystem: of predators and prey—the leopard can change its spots. Information Communications Technology Law. 17, 2, 107–124 (2008).
12. McKeever, S.: Understanding Web content management systems: evolution, lifecycle and market. Industrial Management Data Systems. 103, 9, 686–692 (2003).
13. Meike, M. et al.: Security in Open Source Web Content Management Systems. (2009).
14. Mooney, S.D., Baenziger, P.H.: Extensible open source content management systems and frameworks: a solution for many needs of a bioinformatics group. Briefings in Bioinformatics. 9, 1, 69–74 (2008).
15. K. Patel, Savan; A. Patel, Jayesh; V. Patel Amit. International Journal of Computer Applications, vol. 52, issue 3, pp. 1-5.
16. Raghavan, N., Ravikumar, S.: Content Management System. 1–18 (2008).
17. W3Techs: Usage of content management systems for websites, http://w3techs.com/technologies/overview/content_management/all (2013).
18. Water&Stone: 2011 Open Source CMS Market Share Report. (2011).

Hadoop and its evolving ecosystem

J. Yates Monteith, John D. McGregor, and John E. Ingram

School of Computing
Clemson University
{jymonte,johnmc,jei}@clemson.edu

Abstract. Socio-technical ecosystems are living organisms that grow and shrink, that change velocity, and that split from, or merge with, others. The ecosystems that surround producers of software-intensive products exhibit all of these behaviors. We report on the start of a longitudinal study of the evolution of the Hadoop ecosystem, take a look back over the history of the ecosystem, and describe how we will be observing this ecosystem over the next few months. Our initial observations of the early days of Hadoop's ecosystem showed rapid change. We present these observations and a method for taking and analyzing observations in the future. Our goal is to develop an ecosystem modeling technique that provides practical guidance to strategic decision makers.

1 Introduction

Socio-technical ecosystems are living organisms that grow and shrink, that change velocity, and that split from, or merge with, others. Recently researchers have found it useful to describe the environment surrounding certain software platform-based communities in ecosystem terms. Many of those descriptions focus on the mutual benefit derived from the platform. However, in trying to support strategic decision makers, the true predator-prey notion of an ecosystem, in which both collaborators and competitors interact, gives a comprehensive view of the ecosystem.

Business strategists and software architects both must balance opposing forces to achieve the best possible result for their organization. In an organization that builds software-intensive products the business and technical forces are closely related and interconnected. New business models, such as Platform as a Service (PaaS), require new architectures to accommodate collaborators and to separate that which is the basis for collaboration from that which is the basis for competition. Over time the line between these two shifts as more features become commoditized and organizations innovate to identify new proprietary features. These are the changes that motivate this work.

New algorithms and paradigms are often the basis for new communities and in the early days there is much activity as the forces of competition from established technologies clash with the enthusiasm for the new capabilities. This leads us to some interesting questions:

- How is the ecosystem surrounding a new technology different from that surrounding a mature, established technology?
- What influence does that difference have on the business decisions that must be made?
- Will the frequency and types of change show a different pattern as the technologies mature and the buzz words become accepted termnology?
- How do the linkages between the business and software aspects of the ecosystem respond to changes over time?

In 2004 researchers at Google published a new Map/Reduce algorithm for distributed computation. This algorithm has formed the nucleus of a new ecosystem for distributed computing, which is the focus of this paper.

As pointed out by Hannsen et al, there is a need for more detailed accounts of actual ecosystems and the changes they undergo over time [1]. A portion of our research time is spent tracking a few ecosystems and examining how they are changing. Some data is easy to identify, like major software changes indicated by version numbers; however, most useful data is difficult to identify and parse. Data including both motivations for and changes to code, business models, governance structures, collaborative and coopetitive alliances are all useful data points. By conducting longitudinal studies we have the opportunity to search for patterns in these changes and to anticipate their frequency and direction in the future.

Our current contribution is a baseline report on the Hadoop ecosystem. We apply STREAM, our ecosystem analysis method, to Hadoop distributions from the early releases to the present. We consider two dimensions. We describe portions of the value chain that relates suppliers to customers at the current point in time. We define data useful in evaluating where value is added. We also explore the evolutionary forces responsible for changing where value is accrued over time.

The remainder of this paper is organized as follows: Section 2 and Section 3 provides background information on STREAM; Section 4 describes the observations that were made; Section 5 presents the results collected from the observations; Section 6 provides a view into our future efforts, and Section 7 is a brief conclusion.

2 STREAM

The **STR**ategic **E**cosystem **A**nalysis **M**ethod (STREAM) [2] addresses the various facets of a socio-technical ecosystem which encompasses a community, usually associated through a common interest in a particular domain. STREAM presents the ecosystem through three types of views: business [3], software [4], and innovation [5], which correspond to the three types of ecosystems featured in the ecosystem literature. The business and software views represent the state of the ecosystem at any given moment. The innovation view shows the forces that will result in evolution.

Each application of STREAM is customized to answer specific questions. The exact data collected and the analysis methods applied will directly address those questions.

Each of the types of views has specific attributes, artifacts, and analysis techniques. We introduce each here and give more detail in the case study.

- Business view - The organizations in an ecosystem interact explicitly, e.g. trading partners, and implicitly, e.g. through pricing models. Michael Porter's Five Forces for Strategy Development model gives a structure to this view [6].
- Software view - The software architecture is the major structuring element for the software view. We do as detailed an analysis as possible with the level of architecture description that is available.
- Innovation view - The innovation view represents both business and software innovations. We organize those innovations according to Businessweek's categories of innovation: product, process, business model, and customer experience.

We use the structuring elements in each view to define appropriate abstractions and to guide collection of the data needed to instantiate them.

3 E-STREAM

STREAM as it originally was defined gives a snapshot of an ecosystem at a point in time [2] [7]. E-STREAM is an extension of STREAM that supports modeling the ecosystem's evolution by a combination of a series of snapshots, obtained through multiple applications of STREAM, with measures analyzing the changes in-between the snapshots. In section 4 we illustrate these extensions.

3.1 Ecosystem Evolution

The evolution of both organizations and software have been well studied but the evolution of ecosystems, particularly those that encompass software-intensive products, is not as well understood. Tiwana et al refer to evolution in an ecosystem as coevolution since both organizational and technical changes occur [8]. STREAM handles this coevolution naturally with its multiple views. Tiwana et al proposed a framework for studying evolution of a platform ecosystem that separates "internal" platform forces from "external" platform forces and separates the internal forces into platform governance and platform architecture. The dependency graphs we construct for both organizations and software modules represent this separation and, in fact, allow for multiple separations.

Hanssen et al have conducted a longitudinal study of the ecosystem surrounding CSoft [1] [9] [10] [11] [12]. They hypothesized a set of charactistics for software ecosystems which we will revisit in Section 5.

Evolution is essentially a time-based view of change. Since change often comes about as a result of innovation we have organized the rest of this section using the Businessweek categories of sources of change to discuss evolution. We use forward references into the case study in the next section to illustrate each category.

3.2 Product

Each new release of a software product offers a different value to customers. Some notable exceptions excluded, each release provides more value than the one before. A measure of this value can be seen by looking at the number of releases per year, number of downloads per year, or other measure of use. Our timeline shows releases per year, shown in Figure 1.

3.3 Process

A value chain is a model of the steps through which a product passes as it is created and the value that is added at each step. One paper has hypothesized a value chain for software in which the standard development life cycle phases are the steps in the value chain [13]. In the ecosystem a visible measure of change will be differences in mechanisms by which a product is assembled. For example, the project may begin to provide build processes or pre-configured distributions for targeted groups of developers or users, respectively. The Substitutes section of the Five Forces analysis in Section 4.1 list organizations providing users with improved processes for using Hadoop.

3.4 Business model

Business model changes usually occur as a result of a strategic decision to change directions. Open source projects may maintain repositories of minutes of the governing councils such as the architecture council or a project management committee. Commercial organizations often convert projects which have previously been proprietary to an open source project, e.g. Hadoop.

3.5 Customer experience

The customer experience is tied to the evolution of the business model and the product itself. Defect reports and change requests reflect customer issues and these can be tracked over time. Tools such as Jira allow all users to comment on issues. “Big Data” techniques can be used to mine information from the Jira logs.

4 Case study

Apache Hadoop is a scalable computing framework that abstracts away the issues of data distribution, scheduling, and fault tolerance from applications. Hadoop is a framework that is the core of a rapidly growing ecosystem in which a number of providers are building Platforms as a Service (PaaS) based on Hadoop.

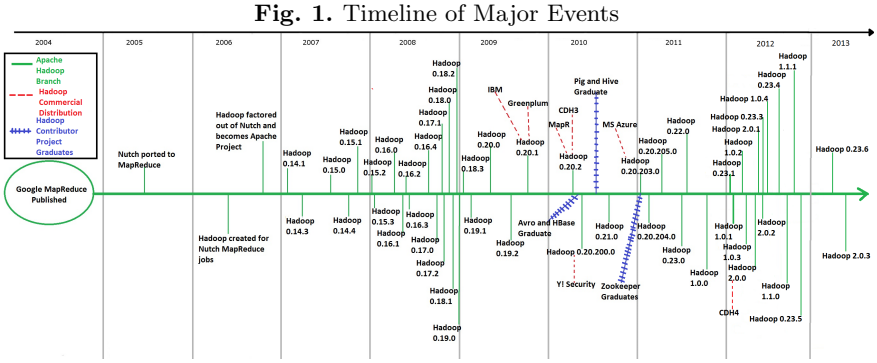
Hadoop utilizes an innovative approach called Map/Reduce intended for processing data collections that are so big that it is more efficient to move the computation to where the data is rather than vice versa. The user of the Map/Reduce

approach writes a Map program that divides the data and directs it to the set of computing nodes. The user then writes Reduce programs that accomplish the needed computation by first computing on each node and then taking the partial result from a node and combining it with the partial results from neighboring nodes to reduce iteratively down to a single answer.

We have developed a timeline, shown in Figure 1 (full color expandable figures can be found at http://www.cs.clemson.edu/sserg/iwseco/2013/), to capture some of the historical information we collected about the Hadoop ecosystem. In 2004 two formative papers were published by Google authors [14][15]. These papers defined the Google file system and Map/Reduce architecture, respectively. Hadoop was initially housed in the Nutch Apache project, but split off to become an independent Apache project in 2006. In January 2010, Google was granted a patent that covers the Map/Reduce algorithm. Three months later Google issued a license to the Apache Software Foundation. Since that time use of Hadoop has grown rapidly.

Over the last few years parts of the original Hadoop Apache project have matured and spun off to become independent Apache projects: Avro, HBase, Hive, Pig, Flume, Sqoop, Oozie, HCatalog and Zookeeper. These products are used with Hadoop depending upon the configuration and are focal parts of the ecosystem.

Fig. 1. Timeline of Major Events



4.1 Business view

At the core of the Hadoop ecosystem is the Apache Hadoop project which maintains the Map/Reduce framework and the Hadoop Distributed File System (HDFS). The project is governed by a project management committee (PMC) that is self-perpetuating and self-directing. Many of the members of the committee are from larger organizations that use the Hadoop distribution as part of strategic product offerings.

Figure 3 shows the network of organizations that contribute to the Hadoop Ecosystem and to which project they contribute. Triangular nodes represent or-

ganizations that contribute personnel to the PMC or committers. The nodes are sized by how many personnel are contributed by the organization to all projects combined. Circular nodes represent the "Hadoopified" projects, their sizes based on how many committers and PMC members they have. Edges between these nodes represent an organization contributing some number of personnel to the project. The edges' thickness is sized to reflect the number of personnel assigned from that organization to that project. In some cases, a single person from a single organization may contribute to multiple projects. This data was collected from the apache.org team-lists for the "Hadoopified" Projects.

Following Porter's Five Forces model we consider the five classes of organizations that influence the direction of Hadoop.

Suppliers Not surprisingly, Hadoop, as an Apache project, mainly pulls from Apache sources. Additional component requirements are satisified by integrating existing open-source and open-license components. Because of the open-source nature of the components, suppliers are unable to leverage profitability of the market into increased profitability; however, greater visibility is useful in convincing organizations to collaborate. Commercial organizations that are contributing code to the Hadoop project, which they have developed to facilitate their proprietary features, are both users and suppliers. We will discuss the supply network in section 4.2.

Substitutes A number of different substitutes for big data analysis are available that diverge from Map/Reduce. GridGrain[16] offers an alternative architecture that also uses a Map/Reduce approach. Rather than use a distributed file system, GridGrain uses an in-memory data grid concept. This architecture handles less data than what is often meant by "big data," hence its classification as a substitute rather than a competitor, but there are many applications for which it is sufficient. Additional substitutes include Spark, ScaleOut, and GraphLab, which offer alternatives to Map/Reduce as well.

Potential Entrants Our analysis did not identify any organizations that are publicly considering entering into this ecosystem.

Competitors The core of Hadoop includes the file system and the computation engine. There are several competitors to the Hadoop file system: Lustre, Orange File System, GIGA+, Google File System, Ceph, and NFile System. Additionally, several alternatives exist to both the Map/Reduce architecture and algorithm, including those offered by Sector/Sphere, Disco, HortonWorks, Cloudera and MapR.

Buyers An open source project has users rather than buyers. Hadoop is used by a large number of organizations. The web-based download makes it impossible to provide a comprehensive list of users. There are some organizations building on top of Hadoop and making their use of Hadoop as a feature. Amazon Elastic, Windows Azure, Google App Engine, and IBM SmartCloud are offering PaaS and IaaS solutions.

There are a number of collaborations being formed around Hadoop that bring organizations together to offer comprehensive configurations that isolate

users from the complexities of replication and fault tolerance. Hewlett Packard, NetApp, and Cisco are a few examples.

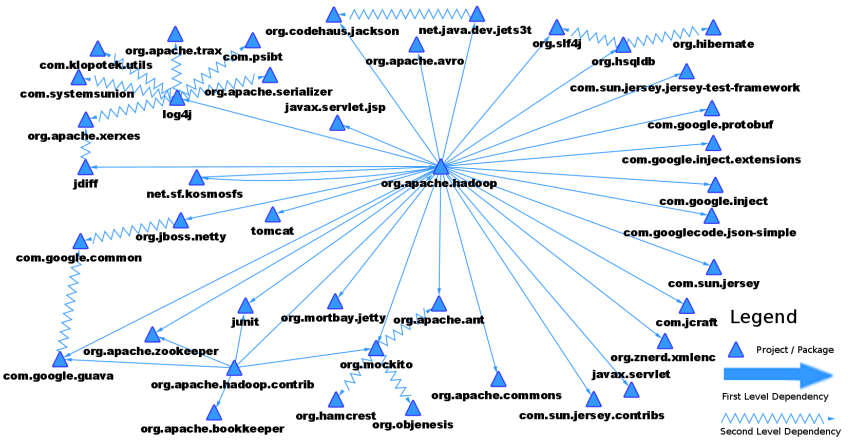
4.2 Software view

The software architecture of Hadoop provides a robust and scalable distributed computing infrastructure for unstructured data.

Although the supply network is made up of organizations, we will consider it from a software product perspective, but we will ignore other suppliers such as makers of development tools. As an open source project the source code is available and shows the references to imported software. The licensing conditions also generally force all imports to be open source and can be used to travel further down the supply network.

Figure 2 shows two levels of the software supply network of Apache Hadoop. The software supply network is modeled as a dependency graph where nodes represent packages of source code, named by their qualified Java package name, and edges represent a "uses" dependency between two source packages.

Fig. 2. Two levels of the Hadoop supply chain



The dependency graph represents the second level of software suppliers, i.e, our suppliers' suppliers. This graph was obtained through analysis of source code for library imports and build dependencies. Nodes connected by a solid line are suppliers obtained by analyzing Hadoop 2.0.3. Nodes connected via zigzag lines represent our suppliers' suppliers.

Due to the fact that many of the third party components included in Hadoop are libraries that facilitate utilities, such as testing, logging or I/O, it is not surprising, though no less interesting, that relationships exist among the software provided by Hadoop's suppliers, which are therefore related implicitly.

4.3 Innovation view

Using the Businessweek categories [17]:

Product The MapReduce architecture was an innovation when Hadoop was initiated. Besides the innovativeness of the architecture, the Hadoop framework uses a functional programming paradigm which is unusual if not innovative. Features such as abstraction of data replication, automatic handling of node failures/fault tolerance and the use of streaming to create a language agnostic interface are innovations in distributed computing products.

Process The concept of storing data and then bringing the computation to it is innovative.

Business model Hadoop itself does not present a new business model, but the other companies that are using Hadoop are implementing high performance computing versions of PaaS and IaaS solutions. These solutions put Hadoop at the core of many products.

Customer experience Hadoop is a fairly traditional open source organization. The evolving architecture that is increasingly modular has made it possible for a customer to replace portions of Hadoop with more hardware specific solutions such as a different file system.

4.4 Risks

The primary risk for the Hadoop project is its current intense popularity and status as a buzz word. The Apache Hadoop core project may have difficulty meeting the needs of the large and diverse number of users. The proliferation of distributions and the independence of the "Hadoopified" projects may lead to divergence. One approach to mitigating this risk would be to broaden the representation on the PMC or to create a separate advisory board that can represent the needs of the diverse user community.

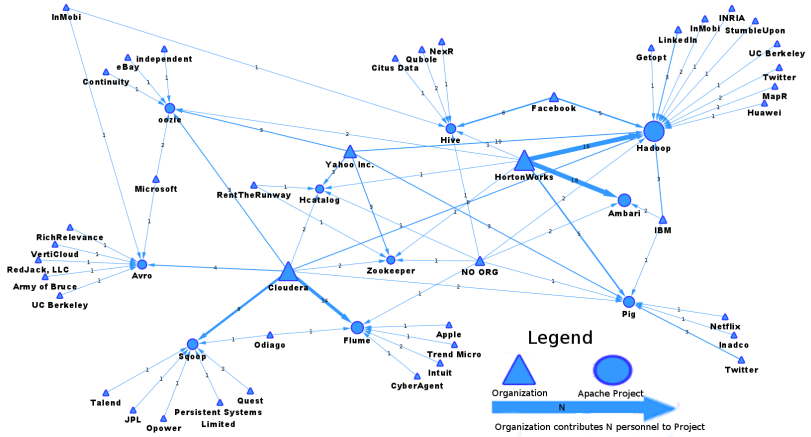
The splitting of several pieces into separate Apache projects potentially harms the architectural integrity of Hadoop. These separate projects are now independent and may make design decisions that will move them away from the trajectory of Hadoop. This is a risk, particularly due to the flat governance structure of Apache projects.

4.5 Ecosystem Health

STREAM uses the ecosystem health criteria described by den Hartigh et al [18].

Robustness In this early phase the ecosystem is very robust. The Project Management Committee (PMC) for Hadoop has representation from several organizations with major resources. The departure of any of these would not cause the project to fail; however, it could cause the project to change directions. Additionally, there are a few dominant players in the Hadoop ecosystem, such as Cloudera and HortonWorks, that support both the core Hadoop project and various "Hadoopified" projects. Figure 3 shows that each of these organizations contributes heavily to projects in the ecosystem.

Fig. 3. Network of Hadoop Contributers



Niche Creation Many organizations are attempting to create niches within the Hadoop ecosystem. "Hadoopified" projects are Apache projects that typically began as Hadoop Contrib projects but have split off into independent projects. These include Avro, Pig, Zookeeper, Flume, HBase, HCatalog, Oozie, Sqoop and Hive. Each of these products delivers a set of features, complementary to those offered by Hadoop, that meets the needs of particular stakeholders. For instance, Hive facilitates data summarization through a domain specific language, HiveQL, that closely mirrors SQL while providing functionality for ad-hoc queries, a useful feature for a database specialist wanting to use Hadoop. Other, mainly commercial, organizations are also working to differentiate themselves from others in the ecosystem. Many organizations are contributing some features to the core Hadoop project and the "Hadoopified" projects. Some organizations create a niche by providing distributions that add end user features making Hadoop easier to deploy, manage, and maintain. Others are providing services related to Hadoop including service and training.

Productivity The ecosystem continues to be very productive. The core Hadoop project maintains four release streams: legacy, stable, beta and alpha. Apache is fixing defects an dreleasing builds. In addition to the efforts that surround HDFS and MapReduce, the set of tools surrounding Hadoop, the Hadoopified projects, represent a significant amount of productivity, with at least nine new Apache projects started since 2008, the majority of which have evolved into top-level Apache projects.

4.6 Evolutionary Forces

In this baseline model we are considering the current state of Hadoop, but we briefly consider the evolutionary forces, both internal and external, at work in the Hadoop ecosystem. Rather than simply internal and external forces there are

layers of forces. At the core there is the Hadoop Apache project with the two fundamental components: MapReduce and the Hadoop file system. Then there are the Hadoopified products which augment, but have split from or formed independently of, Hadoop. Still further removed are those organizations like Cloudera, Hortonworks, and MapR that are adding features to the basic Hadoop distribution, in addition to developing workflow solutions and training and support materials for Hadoop. Further still there are organizations, such as HP and Microsoft, bundling basic Hadoop or a supplemented Hadoop to provided completely configured installations ready for end users who have big data but no systems expertise.

These organizations are addressing several markets segments which present different forces and which will most likely evolve at different rates, new levels may emerge, and organizations may move to different levels. Our longitudinal look will capture these changes.

5 Results

As part of developing this initial snapshot we have we have systematically covered the Apache Hadoop project documentation to identify relevant stakeholders and organizations, visited contributing organizations sites to identify their contributers and analyzed source code for supply-chain modeling in this early baseline model of the Hadoop ecosystem. The data is organized into diagrams which are the information managers use.

Gathering information and conducting analyses on an ecosystem surrounding an open source project has proven to be a difficult, but manageable process. Several observations can be made based on the techniques we have used and the information we have gathered:

- Upstream suppliers can be identified by analyzing build dependencies and library inclusions,
- Niche creation can be evaluated via downstream users who also exist as internal suppliers,
- Productivity can be measured in part by the number and frequency of releases determined from changelogs of projects within the ecosystem, and
- Evolutionary information can be related to a timeline created from the above data.

Hanssen et al hypothesized a set of characteristics for software ecosystems [1]. Our study of Hadoop supports several of those hypotheses:

- central organization - Apache Hadoop project and “Hadoopified” projects provide the basis for the commercial development
- adaptation - decomposing into niche projects to address user needs
- networked - dependencies among software elements
- use of technology - the Apache infrastructure

– shared values - the shared domain leads to certain shared values around distributed computing while there is diversity from a business perspective

While similar tactics might prove useful in ecosystems surrounding commercial or closed source offerings, the abundance of accessible data within an open-source ecosystem is what powers an analyst's ability to model such an ecosystem. It is necessary to continue to gather all available data concerning the emergence or splitting of new projects, the addition of new, or exclusion of existing, software dependencies, the governance structure of projects within the ecosystem and business models leveraging the ecosystem.

6 Future Work

In Section 1, we posed four questions to help guide our analysis of the ecosystem surrounding the Hadoop Map/Reduce framework and architecture. While we are closer to the answers for the questions, the continuing evolution of the ecosystem requires continued data collection and analysis. We will revisit the Hadoop ecosystem at quarterly intervals to add to and revise our models and analyses. Recurring analysis and revision is necessary due to the rapid rate of adoption by companies and the unkowable number of companies using Hadoop, both of which threaten the validity of this study. At these intervals, we will consider more quantified measures for evolution and ecosystem health metrics. Evolutionary metrics may include rates of change on ecosystem elements that were examined in this work: ecosystem size, roles of ecosystem members, external suppliers, release rates and niche offerings. Additionally, the work of the authors in [19] may be helpful in providing project analysis in the software view.

7 Conclusion

We have observed two major trends: a splitting of an initial project into projects that are more narrowly focused and a broadening of the ways in which organizations monetize their participation in the ecosystem. By providing an innovative architecture Hadoop has an advantage but other organizations are already following this approach and offering competing products. STREAM is providing us with a framework within which to add the tools needed to answer the questions in which we are interested. The subsequent snapshots and our analyses of the deltas will provide additional insights about Hadoop specifically and sociotechnical ecosystems in general.

References

1. Hanssen, G.: A longitudinal case study of an emerging software ecosystem: Implications for practice and theory (2012)

2. Chastek, G., McGregor, J.D.: It takes an ecosystem, SSTC (2012)
3. Iansiti, M., Levien, R.: Strategy as ecology strategy as ecology. Harvard Business Review **82**(3) (2004) 6881
4. Messerschmitt, D.G., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press, Cambridge, MA, USA (2003)
5. Adner, R.: Match your innovation strategy to your innovation ecosystem. Harvard Business Review **84**(4) (2006) 98–107; 148
6. Porter, M.E.: The five competitive forces that shape strategy. Harvard Business Review **86**(1) (2008) 78–93, 137
7. Monteith, J.Y., McGregor, J.D.: A three viewpoint model for software ecosystems. In: Proceedings of Software Engineering and Applictions 2012. (2012)
8. Tiwana, A., Konsynski, B., Bush, A.A.: Platform evolution: Coevolution of platform architecture, governance, and nvironmental dynamics (2010)
9. Hanssen, G.K., Faegri, T.E.: Agile customer engagement: a longitudinal qualitative case study. In: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ISESE '06, New York, NY, USA, ACM (2006) 164–173
10. Hanssen, G.K., Fígri, T.E.: Process fusion: An industrial case study on agile software product line engineering. J. Syst. Softw. **81**(6) (June 2008) 843–854
11. Hanssen, G., Yamashita, A.F., Conradi, R., Moonen, L.: Software entropy in agile product evolution. In: Proceedings of the 2010 43rd Hawaii International Conference on System Sciences. HICSS '10, Washington, DC, USA, IEEE Computer Society (2010) 1–10
12. Faegri, T.E., Hanssen, G.K.: Collaboration, process control, and fragility in evolutionary product development. IEEE Softw. **24**(3) (May 2007) 96–104
13. Insemble: Building bridges in the software value chain through enterprise architects "http://www.insemble.com/software-value-chain.html".
14. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles. (2003)
15. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of OSDI'04: Sixth Symposium on Operating System Design and Implementation. (2004)
16. GridGrain: "http://www.gridgain.com/features/".
17. Businessweek: Fifty most innovative companies. (2009) "http://bwnt.businessweek.com/interactive_reports/innovative_50_2009/".
18. den Hartigh, E., Tol, M., Visscher, W.: The health measurement of a business ecosystem. In: Proceeedings of ECCON 2006. (2006)
19. Bjarnason, E., Svensson, R., Regnell, B.: Evidence-based timelines for project retrospectives x2014; a method for assessing requirements engineering in context. In: Empirical Requirements Engineering (EmpiRE), 2012 IEEE Second International Workshop on. (2012) 17–24

Towards the roles and motives of open source software developers

Ruvar Spauwen and Slinger Jansen

Utrecht University
Department of Information and Computing Sciences
Princetonplein 5, 3584 CC, Utrecht, Netherlands
r.a.spauwen@students.uu.nl,s.jansen@cs.uu.nl

Abstract. The software ecosystems of current web browsers include thousands of extensions, which provide additional and customized features for end users and which can generate stronger loyalty towards the browser. Not much research has been done on the development of browser extensions and, in particular, why developers chose to develop for a certain browser. Hence, this research tries to do so by (1) investigating and proposing a set of single platform developer roles and (2) looking more closely at the subset of developers that have developed for multiple platforms, including their behavioral patterns. The selection of browsers consists of Chrome, Firefox, Opera, and Safari, and the researched dataset includes all identified browser extension projects on the web-based hosting service Github between the years 2007 and 2012. The goal of this research is to propose a set of methods that enable clear and meaningful categorization of open source software developers. Consequently, these methods could be seen as stepping stones towards more qualitative-based research, such as: investigating the motives of specific category of developers for contributing to an ecosystem, which can lead to more effective input for controlling or at least influencing an ecosystem's health.

Key words: software ecosystems, internet browser extensions, open source software development, behavioral patterns, developer roles, repository mining

1 Introduction

These days, many companies realize that the total value of a software product can not be defined only by the value of the product itself. Instead, the total value of a software product can be better defined as the sum of a product's value and the values of other products that are interacting with the specific software product [17]. This creates a network of mutual dependencies in which products, companies and services work together to keep the network alive, and such type of network was first described by Messerschmitt and Szyperski in 2003 [14] as a software ecosystem (SECO). In 2009, Jansen, Finkelstein and Brinkkemper proceeded by defining 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" [10]. Free and Open source software (FOSS) projects tend to have a SECO that forms itself around the projects in which their growth is determined by the amount of developers that are willing to contribute to projects and create new projects [12]. Companies could provide support by offering toolkits and developer platforms, but the success of these types of software projects still depends on the effort of the developers themselves. This research project focusses on free and publicly available software development that is liberally licensed to grant users the right to use, copy, study, change and improve the source code [4].

An important question concerning FOSS development relates to why developers participate in these projects. Research has shown several motives for participation, for instance: "pleasure", "a form of personal rewarding" and "an opportunity to improve technical skills" [2,16]. Other research names reasons like "building trust and reputation", "showing creativity", "advancement through increasingly challenging technical roles" [11], as well as "being generous in sharing time expertise and source code" [1]. Additionally, it often occurs that FOSS developers are involved in several projects: a study of 2002 showed that 5% of the examined FOSS developers were involved in 10 or more projects [6]. Madey et al. [13] examined the importance of these so-called "hubs" or "linchpins", through social network analysis, and showed that the absence of these developers can result in the segregation of networks. However, these studies do not specify if developers are related to one or several platforms and, more specifically, whether developer perform the same role in every SECO or that they might perform different kind of roles.

To the authors' knowledge, in-depth research on the behaviour of these multiple platform developers does not exist. Idu, van de Zande, and Jansen [9] performed an interesting case study on Apple's App Store ecosystem, but their scope is more focused on a single company with multiple sub-ecosystems. On the other hand, Hyrynsalmi et al. [7] do consider the SECOs from different mobile software companies and discuss characteristics of multiple platform developers. Only their research does not elaborate on how developers might behave over time. Therefore, this research attempts to go a step further by using the developer behaviour perspective over time to work towards, possibly valuable, motives of developers for participating in certain FOSS projects.

More specifically, knowledge of behavioral patterns might be useful in a way that it can lead to new methods for influencing a SECO's health; i.e. information about developers' relationships with platforms and other developers over time can be used as an indicator of the robustness of a SECO, which is one of the determinants of measuring SECO health [8]. Also, it could be used to define group stability, connectedness and outbound links, which are part of the metrics Den Hartigh defines to measure robustness [3]. If companies have access to developers' motives, for instance, for choosing to contribute to a specific platform or not, they can use this knowledge for more accurate improvements to their platform, which in return might lead to stronger relationship with current developers as well as more future developers.

Finally, the accessibility of source code, alterations over time and information about contributing developers make FOSS projects suitable for (dynamic) network analysis and they provide new ways of exploring large software ecosystem, as shown by [12]. Additionally, the large number of FOSS projects that are being hosted on open source software hosting websites like Github, Sourceforge and Google Code, increase the reliability of statistical analysis results [18].

The remainder of this paper is organized as follows. In the following section, we present the main research questions and their related sub questions. Subsequently, the research method is discussed in section 3, providing argumentation on the chosen type of research, the selection of cases and collection of data. Section 4 contains the analysis on the gathered dataset and the following results are presented in section 5. Finally, we conclude this paper by giving an overview of the things discussed, point out important limitations and propose several opportunities for future work.

2 Research Questions

The main research questions answered in this paper is: **"How can we divide open source developers of browser extensions into distinctive categories and roles?"**. From the main research question, the following sub questions are derived:

1. **What are the characteristics on which we can categorize developers in the separate browser ecosystems?** - By looking closely at the developers and their activities per platform, we will be able to extract the most important characteristics on which the developers can be categorised.
2. **How can we combine the single platform developer characteristics into roles** - In order to get a better overview of the distributions of characteristics per browser, we have to combine them into meaningful roles.
3. **What are the combinations in which developers are or have been connected to more than one ecosystem?** - Before we can research multiple platform developers and their behaviours, we first need to locate them and determine if different behavioral patterns exist.
4. **How can we quantify the different types of multiple platform developers?** - We need to design a model and visual representation that can give meaning to the discovered behavioral patterns and that enables clear and valuable categorization of multiple platform developers.

3 Research Method

In the following sections, we elaborate on the applied research method for gathering the data and performing the analysis. This includes: descriptions of the selected cases and the data source, explanation on how we performed the actual data gathering and on our attempts on ensuring the validity of the research.

3.1 Case selection and description

The selection of cases for the research consisted of the following three criteria. First, the cases had to be part of the same type of software ecosystem to be able to compare them. Second, access to case related software development projects had to be available (e.g. source code, developers and change history) and, finally, these projects needed to be recognizable during the data mining process.

The selection of cases consist of the following web browsers: Chrome (Google), Firefox (Mozilla), Opera (ASA) and Safari (Apple). The reason why Internet Explorer is not selected, will be clarified in the following section. Browser extensions are small software programs that can modify and enhance the functionality of the browser and can be created by internal (e.g. the browser company) and external developers, like commercial companies or (collaborating) independent developers. Unfortunately, there is no current and complete overview of the total number of available extensions per browser and, although all browser and their extensions are freely available and the browser manufacturers actively support FOSS development, not all extensions have public source code. Therefore, to ensure objectiveness and completeness of the research, the collection of extensions is narrowed down to only those that are developed and stored online in Github repositories. Github is a website that provides source code hosting for repositories that use the Git revision control system and provides several social networking features to improve collaboration among developers. Additionally, the website offers both paid plans for private repositories as well as free accounts for FOSS projects. As mentioned in section 1, alternatives to Github and the Git revision control system exist, but due to the time scope of this project and the differences in systems (e.g. what kind of data is stored per repository), this research only includes the most popular combination [5, 15]. Furthermore, Github provides the necessary structure to be able to divide the dataset into extensions, unique developers and commits, including details about the size and the time of the commit and its author. Consequently, the following relationships among the entities are ensured: developers can create multiple commits for multiple extensions and an extension can be related to one or more developers.

3.2 Data Gathering

In order to be able to perform the analysis, quantitative data needed to be collected for each of the cases. Although Github provides access to sufficient data, the repositories are stored in such a manner that they can not be categorised instantly. For instance: the search-string “Firefox extension” produces many false-positives, because the search function of Github includes a repository description field, which often consists of misleading data (e.g. “this is an alternative to the Firefox extension”). Fortunately and in contrary to Internet Explorer, each of the selected browsers have specified a so called “manifest” file, that needs to be included in order to make the extension executable for the browser. Table 1 provides an overview of the manifest files considered for determining if a repository contains a manifest file. Unfortunately, in practice

it appeared that not all repositories have included a manifest file (e.g. yet, anymore or never had one). Also, it appeared that older versions may have used different manifest file types (e.g. Firefox), and that for some platforms only the file type and not the combination of filename and type is mandatory. The last case might increase the chance for false-positives if a file type is also used in other programs than browser extensions. Therefore, when we considered a possible manifest file, we also looked at values inside the file, as can be seen in the third column of Table 1. Because not all of these values are specified by the browser manufacturers as being mandatory, a manifest file was considered valid if and only if it contained at least two of its related values. The manifest files that have no values specified were considered as special cases, for instance: if the scraper script did not find a Firefox "install.rdf" or "manifest.json" file but it did find a "chrome.manifest",".manifest" or ".xpi" file, then the repository would be checked manually.

Platform	**Files**	**Values**
Chrome	manifest.json	"name", "version", "description"
Firefox	install.rdf	"xlmns:em=http://www.mozilla.org/2004/em-rdf#"
		"<em:id>", "<em:version>", "<em:name>", "<em:description>"
	chrome.manifest	n/a
	manifest.json	"name", "version", "description", "author"
	.manifest/.xpi	n/a
Opera	config.xml	"xlmns=http://www.w3.org/ns/widgets", "<name", "<name>", "<description>", "<author>"
Safari	*.plist	"<plist>", "<plist", "<dict>", "<key>"
	*.safariextz	n/a

Table 1: Description of manifest files and related values per platform

The list of candidate repositories was created by searching on Github with different combinations of keywords, like: "Chrome extension 2011-01-01...2011-12-31". Because there is no consensus on the term "extension", alternatives like "add(-)on" and "plug(-)in" were considered as well and subsequently combined with the four platforms and different date intervals. To clarify, a date interval relates to those repositories that were created in that specific period. It was necessary to include this because Github only returns a thousand repositories per search query. Additionally, only repositories that are no Forks of other repositories were considered, because these repositories would skew the data since it is impossible to combine their commits into unique contributions. Finally, due to the design of Github it was decided to count a commit in a repository containing extension for multiple platforms (e.g. Chrome and Firefox) for both platforms.

Different methods are used for collecting the actual data. Due to the extensive checks for manifest files, it was not possible to use Github's developer API for the initial filtering of repositories and so HTML scraping methods were used for this. When this was finished, we were able to use the quicker Github API to retrieve a vast amount of additional information on the developers and commits.

4 Analysis

This section provides the analysis performed on the dataset retrieved from Github, so each question posed in section 2 can be assessed. But first, we provide a description on how the final dataset was established.

The combinations of keywords used to search on Github produced a result of more than 30.000 separate repositories of which almost 9.000 passed one of the platform specific manifest-file checks. Next, this number was reduced to 7.749 extensions from 7.564 unique repositories, by first automatically removing the extensions that had no commits in the period of interest and the repositories that were incomplete. The period of interest was set between 2007 and 2012, because there were only a few (Firefox) manifest-containing repositories before that period. In Chapter 6 we argue that this selection, which is also implemented in the metrics and normalization, can be improved due to the fact that the other three browsers started supporting extensions in 2009 and 2010, which is more than five years after Firefox. Then, a second filter was applied on the dataset by manually controlling an arbitrary selection of outstanding cases, like: odd date values (e.g. "1/1/1970"), extremely large commits (e.g. bulk import) and the default developer accounts "invalid-email-address", "unknown" and "(no author)". Finally, the remaining set of extensions was as follows: 4.746 (Chrome), 2.032 (Firefox), 772 (Safari) and 199 (Opera). Furthermore, more than 340.000 commits and nearly 10.000 developers were collected. The performed analysis is mostly focused on relative values because there is no complete information on the total number of extensions outside of Github and, despite all measures during the gathering of the data, it is not certain whether every record is a finalized extension (i.e. available at one of the official extension markets). Fortunately, the obtained dataset is sufficiently large for dividing developers into different categories and for observing relationships among developers and different platforms.

4.1 Single Platform Developer Roles

This sections explains the design of the single platform developer roles and includes an overview of the distributions per platform. The roles are based on a set of metrics which are aggregated into the following three scores: **T**, **N** and **C**. We argue that a valuable method to categorize developers is provided by a combination of the developer's connectedness with other developers and extensions (**N**), the frequency and intervals over which the developer creates commits (**T**) and the number and size of these commits (**C**). The base metrics are inspired on metrics from related literature, but they are modified and combined in such a manner that further research is necessary to validate their design and the results.

The three scores range from 0 to 1 and they can be visually represented by a 3-dimensional graph divided into eight same-sized cubes, or "octants". Each one of these cubes has distinctive properties that can be related to a specific role. Table 2 gives an overview of these eight roles, including their label and specific properties. The first property of the roles relates to the score **T**, where the value *Occasional* or *Regular* is selected depending on a developer's average

contribution time per extension combined with the total number of days between a developer's first and last commit. Next, the scores **N** and **C** relate to the second property of the role, where specific combinations of scores lead to the values *Adjuster*, *Investor*, *Networker* or *Collaborator*. For instance, the values *Adjuster* and *Investor* are selected when the developer has a low **N** score combined with a low, respectively high **C** score, and the values *Networker* and *Collaborator* are assigned when the **N** is high and the score **C** is low, respectively high.

Labels	**Roles**	**T**	**C**	**N**
a	Occasional Adjuster	0,0 - 0,5	0,0 - 0,5	0,0 - 0,5
b	Occasional Investor	0,0 - 0,5	0,5 - 1,0	0,0 - 0,5
c	Occasional Networker	0,0 - 0,5	0,0 - 0,5	0,5 - 1,0
d	Occasional Collaborator	0,0 - 0,5	0,5 - 1,0	0,5 - 1,0
e	Regular Adjuster	0,5 - 1,0	0,0 - 0,5	0,0 - 0,5
f	Regular Investor	0,5 - 1,0	0,5 - 1,0	0,0 - 0,5
g	Regular Networker	0,5 - 1,0	0,0 - 0,5	0,5 - 1,0
h	Regular Collaborator	0,5 - 1,0	0,5 - 1,0	0,5 - 1,0

Table 2: Overview of roles and related scores

After defining the roles and calculating the scores, we selected for each platform the top 10 percent of developers having the highest combined **T**, **N** and **C** scores. Despite the significant differences in the resulting number of developers per platform (e.g. 576, 357, 21 & 98), these subsets provide the most interesting developers and a correct comparison is ensured by selecting the same relative number of developers for each platform. Next, the developers were given a role based on their **T**, **N** and **C** scores and the resulting distributions are shown in Figure 1. It can be seen that Chrome, Firefox and Safari appear to be surprisingly similar: they all have the *Occasional Adjuster* as the most occurring role, followed by the *Occasional Networker*, the *Regular Adjuster* and a small portion is taken by the *Regular Networker* or *Occasional Collaborator* roles. Regarding Opera, we can see that by far the largest part is occupied by the *Occasional Networker* role. This might be caused by the fact that many of the Opera extensions in our dataset share their repository with extensions for other platforms and this results in a more than average number of connections with other developers.

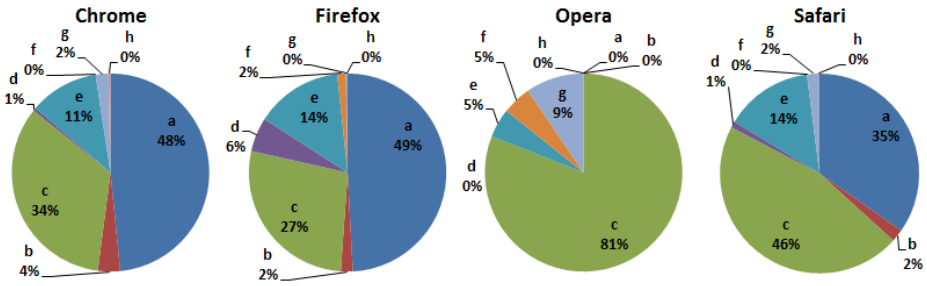


Fig. 1: Top 10 percent distribution of single platform developer roles per platform

4.2 Multiple platform Developers

The second part of the research only considers the subset of multiple platform developers, or MPDs, which include the developers that have contributed to different extensions for more than one type of browser platform, to a single repository that contains extensions for more than one type of browser platform, or both. The dataset contains a total of 743 MPDs of which there are 629, 69 and 45 developers having a connections with 2, 3 and 4 different platforms respectively. An overview of the different combinations and their occurrences can be seen in Table 3. It is clear that the combination Chrome-Firefox is the most occurring, but we can also see that the combination Chrome-Safari is quite significant with 136 developers. It is clear that combinations of platforms containing Opera occur the least often, but percentage-wise, the set of Opera developers consist of the largest number of multiple platform developers (38%) compared to Chrome (12%), Firefox (16%) and Safari (31%).

Platform	**Combinations**										
Chrome	*	*	*		*	*	*				*
Firefox	*		*	*	*		*	*		*	
Opera					*	*	*	*	*	*	*
Safari		*	*	*	*				*	*	*
#	409	136	59	59	45	19	7	3	3	2	1

Table 3: Number of developers per multiple platform combination

A quadrant-based model was designed that rates developers upon the following two criteria: *Fluctuation over Time Active* and *Amount of Contribution*. The first criterium relates to how often a developer switches to another platform or a combination of platforms: this is a combined score that looks at the fluctuation per month, fluctuation per year and the total length of activity (e.g. first and last commit) of the developer during the previously mentioned main scope of this research. Two different time interval levels were considered, because: analysis showed that not every developer creates a commit every month, but if you would only look at intervals per year the granularity would be too low to discover valuable fluctuations. The total time of activity of a developer is included, so that fluctuations of developers that have a longer activity time are weighed heavier. The second criteria relates to the number of commits combined with the size of these commits, regarding number of additions and deletions. A logarithmic normalization was used on the number of additions and deletions, because the dataset contains a small collection of high outliers which often consist of less valuable bulk import commits, and it enables a better distinction between the large number of developers with relatively low "addition" and "deletion" values. The metrics used in this model are partly based on existing metrics, but the combination of them and output of results are based mostly on instinct and have not yet been extensively validated. Therefore, the analysis presented in section 5.2 is solely meant for giving an indication of the model's possibilities.

5 Results

The following sections elaborate upon the results in the context of the research questions posed in section 2. The first two sub questions are depicted in section 5.1 and section 5.2 elaborates on the findings related to the remaining two.

5.1 Single Platform Developer Roles

The distributions of the developer roles, introduced in section 4.1, provide an overview of certain properties of developers and their occurrences as well as similarities and differences between platforms. Figure 1 shows that most Chrome, Firefox and Safari developers score low on the length and size of their contributions. Furthermore we can see that for both Safari and Opera the most occuring role is the *Occasional Networker*. But as mentioned in section 4.1, we believe that the set of Opera developers is skewed because a significantly less amount of repositories containing only an Opera extension is present in the dataset. The reason for this is uncertain and therefore interesting to further investigate upon.

5.2 Multiple Platform Developers

The model introduced in section 4.2 has been applied to the collection of multiple platform developers in the dataset. A graphical representation of the results can be seen in Figure 2. To ensure the clearness of the graph, the developers are grouped based on similar ranges (e.g. $0,1 < x \leq 0,2$ and $0,5 < y \leq 0,6$) and represented as a bubble in the graph, with a size relative to the number of developers in the group. Additionally, the bubbles that contain more than one developers have a label showing the number of developers inside them and, for explanatory purposes, IDs of several outstanding developers have been included.

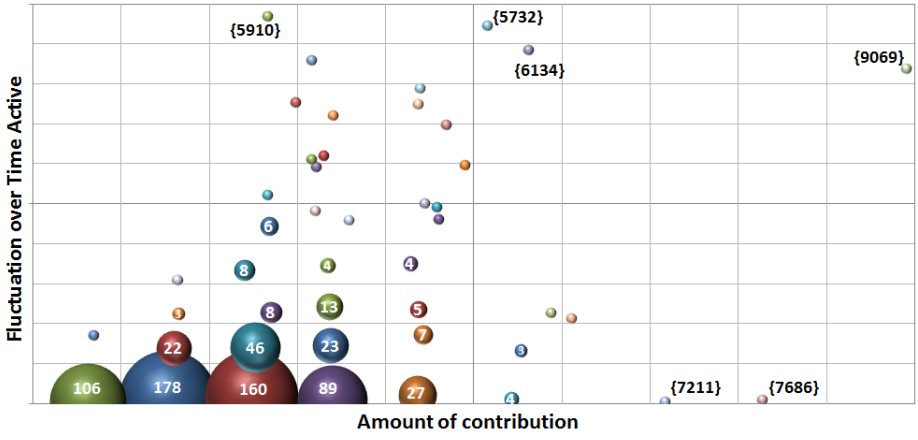


Fig. 2: Distribution of the multiple platform developers

Figure 2 shows that most developers score low on both criteria (e.g. located in the bottom left quadrant). This implies that these developers have a relatively low contribution combined with a stable loyalty to a certain combination of platforms. More interesting are the developers positioned in the other quadrants. For instance, if we look more closely at the three most right developers, named "jcutler", "2d1" and "johnjbarton". Table 4 shows that "johnjbarton" has a average number of commits, but his total number of additions and deletions is significantly higher than the other two and therefore his *Amount of Contribution*, despite the logarithmic normalization, is valued higher. The table shows also that on the level of **Years**, both "jcutler" and "2d1" did not change their combination of platforms. Developer "johnjbarton" does have fluctuations on the level of **Years**, namely: between 2007 and 2010 he developed only for Firefox, then from 2011 both for Firefox and Chrome and, finally, in 2012 only for Chrome. More specific, it shows that 2011 is divided into two periods, as the **Months** value also shows: until July he developed for Firefox and later on only for Chrome. Given his high scores on both criteria, it would be interesting to investigate why "johnjbarton" switched to Chrome after four years of loyalty towards Firefox.

Developer		**Fluctuation over Time**			**Commits**		
ID	**Name**	**Years**	**Months**	**Time (d)**	**Commits**	**Additions**	**Deletions**
5732	jrburke	2	3	2017	2087	1126540	264572
5910	georgebrock	4	3	82	15236	263	79431
6134	Ajnasz	2	6	1699	3805	91528	109521
7211	jcutler	0	0	650	2087	609714	425510
7686	2d1	0	2	286	6957	110735	21584
9069	johnjbarton	2	1	1933	3805	3918354	2095710

Table 4: Overview of multiple platform developers from right quadrants

Next, some remarks can be made regarding the top three developers from Table 4, namely: {5732}, {5733} and {6134}. These IDs relate to the developers named "jrburke", "slightlyoff" and "Ajnasz" respectively, and they are part of the developers with the highest *Fluctuation over Time Active* values combined with significantly high *Amount of Contribution* values. They all have been active developers for at least 5 years and they all have created a commit in the last month of 2012. Furthermore, although they have only developed for the Chrome and Firefox platform, they are all related to multiple different extensions. For instance, "jrburke" has created commits for a total of 8 different extensions. Another remarkable observation from the dataset, is that the developer "slightlyoff" was not active during the years 2009 and 2010 and when he started developing again, his loyalty changed from Chrome to Firefox. Regarding the developer "Ajnasz", we observed that his main loyalty is towards Firefox, but we saw that in 2010 and in 2011, during different periods, he created commits for Chrome extensions, but none of these periods lasted longer than one month. This behaviour is quite different than the behaviour found at the other mentioned developers.

6 Discussion

First of all, the amount of collected research data and the number of sources can be increased: as mentioned in section 3 only repositories hosted on Github have been considered, while there are several other sources (e.g. SourceForge, GoogleCode and CodePlex). Furthermore, the dataset could be extended within the context of Github: although multiple combinations of keywords have been used, it is possible that not all significant extensions have been retrieved. For example, names and documentation in non-English languages could withhold extensions from being discovered; Also, plural forms of keywords could be included and a list of extensions names available on each of the official Extension web stores could be created, although there are some additional challenges to this method (e.g. differences in names between project development name and web store version). To improve the completeness of the dataset and results, extension for Microsoft Internet Explorer, the second most used web browser in the world, should be included. If this was possible it would provide additional comparison materials and most likely more multiple platform developers.

Further improvements can be made regarding the contents of the current dataset. For example, when looking at repositories containing multiple extensions for the same platform: currently, a maximum of one extension per repository per platform combination is considered. Accepting more extensions would greatly increase the data gathering time, because then the scraper script has to consider all the directories of a repository. Another issue is that we are not certain whether all extensions in the dataset are actual finished extension. An effective method for validating all the extensions with the browsers' official markets is needed to improve the ratio of installable extensions in the dataset. One can argue that unfinished, failed or test projects are some kind of contribution to a certain platform, but it would be wise to additionally investigate manifest-containing repositories and verify that they are truly related to a web browser.

7 Conclusion

As discussed, this research is mostly intended as a step towards qualitative research: due to designs of the data source and type of SECOs, many validity issues remain which negatively influence quantitative analysis. Besides providing extensive insight into these issues, the dataset was analysed in such a manner that the results could lead the way to succeeding qualitative research. This was done by suggesting different developer roles and how these were distributed per platform. For the group of multiple platform developers we presented an overview of platform combinations, a quadrant based on specific criteria and we discussed several outstanding examples of developers and their behavior patterns. Based on these examples, we proposed several questions on which future work can be based on. Additionally, the following questions can be considered when including the single platform roles: "How do the single platform roles of a multiple platform developer compare to each other?" and "Does a developer changes roles if you

look at specific periods?". These questions should be interpreted as guidelines: guidelines on how roles, relationships among developers and behavioral patterns could be used to extract the motives of certain developers for participating in and contributing to certain development projects. We hope that these guidelines lead to more reliable methods for influencing the health of open source SECOs.

References

1. M Bergquist and J Ljungberg. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 11(4):305–320, 2001.
2. K Crowston and B Scozzi. Open source software projects as virtual organisations: competency rallying for software development. *IEE Proceedings - Software*, 149(1):3–17, February 2002.
3. E Den Hartigh, M Tol, and W Visscher. The Health Measurement of a Business Ecosystem. *Ecosystems*, 2783565:1–39, 2006.
4. J Feller and B Fitzgerald. *Understanding open source software development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
5. K Finley. Github Has Surpassed Sourceforge and Google Code in Popularity. `http://readwrite.com/2011/06/02/github-has-passed-sourceforge`, 2011.
6. A Hars and S Ou. Working for free? Motivations of participating in open source projects. *International Journal of Electronic Commerce*, 6(3):25–39, 2002.
7. S Hyrynsalmi, T Mäkilä, A Järvi, A Suominen, M Seppänen, and T Knuutila. App Store, Marketplace, Play! An Analysis of Multi-Homing in Mobile SECOs. In *Proc. 4th Intern. Workshops on SECOs*, volume 879, pages 59–72. CEUR, 2012.
8. M Iansiti and R Levien. Keystones and Dominators: Framing the Operational Dynamics of Business Ecosystems. page 84, 2004.
9. A Idu, T Van De Zande, and S Jansen. Multi-homing in the Apple ecosystem: Why and how developers target multiple Apple App Stores. In *Proc. of the Intern. Conf. on Management of Emergent Digital EcoSystems*, pages 122–128. ACM Press, 2011.
10. S Jansen, A Finkelstein, and S Brinkkemper. A Sense of Community: A Research Agenda for Software Ecosystems. *31st International Conference on Software Engineering Companion Volume*, (C):187–190, 2009.
11. C Jensen and W Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings of the 29th international conference on Software Engineering*, pages 364–374, 2007.
12. J Kabbedijk and S Jansen. Steering insight: An exploration of the ruby software ecosystem. In B Regnell, I Weerd, and O Troyer, editors, *Software Business*, volume 80, pages 44–55. Springer Berlin Heidelberg, 2011.
13. G Madey, V Freeh, and R Tynan. *Free/Open Source Software Development*. IGI Global, July 2004.
14. D Messerschmitt and C Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*, volume 1. The MIT Press, 2003.
15. I Skerrett. Eclipse Community Survey Result for 2012. `http://ianskerrett.wordpress.com/2012/06/08/`.
16. G Von Krogh, S Spaeth, and K Lakhani. Community, joining, and specialization in open source software innovation. *Research Policy*, (7):1217–1241.
17. L Xu and S Brinkkemper. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, 2007.
18. RK Yin. *Case Study Research: Design and Methods*, volume 5. Sage Pub., 2009.

Notes from the IWSECO 2013 panel discussion

RTn = Research Take Aways:

RT1: Relationship SW Architecture vs SECO ==> Clopeness ==> IP

RT2: Operationalizing Health Metrics ==> Context ==> Not viewing SECO just for health sake. Compare it to requirements. At the same time, should also evaluate temporal health measures. How is it doing currently and how will it in the future?

RT3: More longitudinal observational studies & more empirical works RT4: How to create an ecosystem RT5: Become more relevant to industry RT6: What can we learn from other ecosystems like manufacturing or social? RT7: Are standards fundamental to SECO success?

Jan Bosch: The reason why it was difficult to create an ecosystem: we did not provide a business model for the ecosystem. The discussion was: 'why would I give away hundreds of millions of revenue to other people.' The answer is: 'why do people still submit apps to the App Store - there is a business model. Creating an ecosystem is simple. Throw money at it!'

Sjaak Brinkkemper: But what are these business models?

Pasi Tyrvanen: And how much money should we throw at it? Ecosystem businesses do not necessarily start working with a parent company through subsidies.

Slinger Jansen: There is a really strong call for theory, developing theoretical models behind software ecosystems. There should be more empirical studies. We should be doing studies into OSS, commercial organisations, repository mining (re-use, dependencies, health). Go out there and collect the data. The larger your scope the better. The commercial side of things - we need data. Microsoft will give you a list of ecopartners, but will not tell you where the money is.

Carina: There is a need for good, reliable research methods & approaches. Who can replicate your own study.

Slinger Jansen: I think many people are interested in how you govern an ecosystem. What are the findings from the studies that we did. Should an ecosystem have more lone wolfs, groups or new people? What is beneficial to the ecosystem?

Geir Hanssen: I think one challenge is, becoming more relevant for industry. My motive lies within being helpful to industry. I see in the RT-list that there are some answers. We have spent some years on defining and understanding; we are entering the phase of recommending industry.

Jan Bosch: Several people here mentioned that they have a company, customers and want to create an ecosystem. But how do I do it? Before we can recommend industry, we should build our own ecosystem. I think the prescriptive part is difficult because the moment I find out, I am not going to tell you.

Pasi: Title for the next ICSOB: 'How to create an ecosystem?' Slinger & Jan: We could at least write a paper about how not to do it. Lots of data.

Jan Bosch: ‘What would we advise NOKIA to be successful in the Windows 8 ecosystem?’ Pasi, what do you do?

Pasi: I do not as an academic.

Jan Bosch: ‘Another interesting question is, what ecosystem do I join to create a successful business?’

Sjaak: The issues in the SECO subject are similar to automotive or aerospace. What can we learn from the manufacturing ecosystem? What I observe is that technical standards are very important for the survival of a business. (You can buy everything, as long as it is speci!ed). I am sure that over the years many different standards popped up, and at some point consolidated. Would also standardization of interfaces be a success determination of ecosystem. This is my API, please build as many $THING as you like.

Jan Bosch: I agree that standards are useful in ecosystems, but you want to strategically standardize. Sometimes you do and sometimes you disrupt standards to give other companies a strategic disadvantage. The moment you standardize it becomes hard to innovate over that standard.

Other communities we can learn from are social communities in the broadest sense. Facebook, Twitter, etcetera - they have found effective ways to get (software) ecosystems going. It is not the money but reputation or other forms of value. How do we implement that in software ecosystems?

Negative and positive impressions of the workshop

Positive	**Negative**
Atmosphere	More participation
Topical discussion	Need for interaction
Focus on results	More ecosystem stuff
Close knit community	More provocation
Utrecht	Beamer!!
Keynote	More about people
Great Community	Location
Business vs Social	More empirical
	Little time