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.

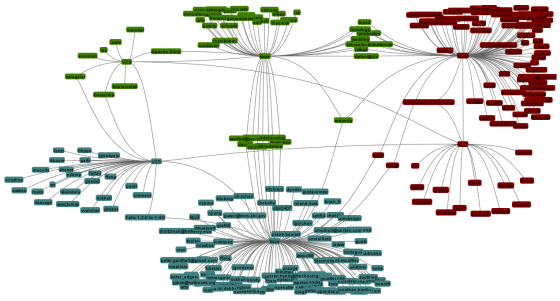


Fig. 1: Communities View, featuring Apache Sling (green), Apache Jackrabbit (red) and Apache Felix (blue) projects.

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.

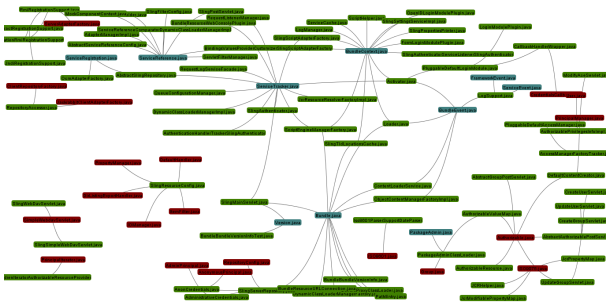


Fig. 2: Technical View presenting dependencies between Apache Slings artifacts (green) and Apache Felix (blue) and Apache Jackrabbit (red) projects.

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.

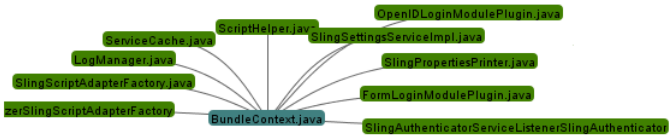
5.4 Results

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.

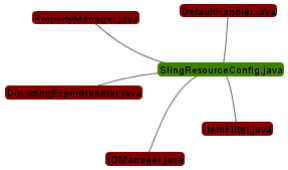
Technical Discussion For the technical perspective analysis we focused on the dependencies between Apache Sling and both Apache Felix and Jackrabbits, excluding dependencies between the latter projects. We did that since the graph that resulted when considering all projects was too dense and proved difficult 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