The Co-evolution of Socio-technical Structures in Sustainable Software Development: Lessons from the Open Source Software Communities

Marcelo Serrano Zanetti
ETH Zurich
mzanetti@ethz.ch

***Abstract*—Software development depends on many factors, including technical, human and social aspects. Due to the complexity of this dependence, a unifying framework must be defined and for this purpose we adopt the complex networks methodology. We use a data-driven approach based on a large collection of open source software projects extracted from online project development platforms. The preliminary results presented in this article reveal that the network perspective yields key insights into the sustainability of software development.**

***Keywords*-complex networks; statistical physics; social networks; software dependency graphs; open source software; free software; quantitative analysis; mining software repositories**

I. Introduction

The open source way of software development has been the focus of a considerable amount of academic research [1], not to mention the commercial interest evolving around it. Indeed, in some cases, open source software (OSS) can compete with or even outperform its commercial competitors; the APACHE HTTP server being a good example for such a success of OSS development.

OSS is an interesting research topic because it occurs at a global scale and the vast majority of the information flow goes through the Internet. Therefore, given suitable data mining tools, online data can be used in empirical research covering most of the aspects involved in OSS development. Available data range from the whole source code history stored in code repositories and project development platforms (e.g. SOURCEFORGE and more recently GITHUB), to the actual interaction between developers and users available in the form of discussion forums and mailing lists.

In order to understand the processes shaping OSS, it is necessary to bridge complementary perspectives of multiple disciplines [2]. Social sciences are needed in order to understand interactions between developers and users, management and economic sciences provide insights into the organizational structure of projects and software engineering allows us –among other things– to assess the impact of technical aspects like design patterns, programming languages and paradigms.

A solid understanding of how the interplay of these processes hinders or fosters sustainable software engineering is relevant for software projects in general. Therefore, the impact of interdisciplinary research on the dynamics of OSS projects is believed to stretch out far beyond the OSS domain. However, in order to study the interrelation of technical, social and organizational facets of software engineering in a quantitative way, one first needs to be able to reframe them in a unifying framework. An interesting aspect of OSS is that the communities of users and contributors as well as the structural features of software architectures can be studied from a network perspective. Due to recent advances of complex networks methods in assessing the structural and dynamical features of networks with complex structures, we argue that the resulting set of tools and abstractions is a good candidate for this task. As of today, the complex networks framework is a well established methodology and has been successfully applied in multiple research fields, including the social sciences, technical systems, biology and economics (see e.g. [3], [4]).

There are research works addressing the network nature of different aspects of software development separately, usually focusing on very specific research questions and applications, for example the automatic detection of developer roles within a project [5] or the evolution of dependencies and conflicts between software components [6]. However, to the best of our knowledge, the relations between and co-evolution of networks representing different aspects of software projects have not been studied from this unifying perspective. Therefore the main goal of our research is to add this perspective to the current understanding of the co-evolution of the socio-technical structures found in software development processes.

By pursuing this goal, we want to contribute not only to the OSS community, but also to the other disciplines involved. More specifically, we aim at

- studying the impact of programming languages and programming paradigms on dependency structures and co-change (chains of changes triggered by punctual modifications) dynamics.
- proposing metrics capturing features of sustainable software architectures which may be implemented in tools aiding distributed software development.
- contributing to physics of complex networks by enriching existing network growth models with domain knowledge inspired by the dynamics observed in OSS.
- measuring the impact of developer turnover (replacement) on the OSS development process.

978-1-4673-1067-3/12/$31.00 © 2012 IEEE

ICSE 2012, Zurich, Switzerland
Doctoral Symposium

- understanding the dynamical interdependence and interaction (co-evolution) of socio-technical networks found in OSS communities.
- predicting how the aforementioned issues result on success or failure of OSS projects and their communities, by defining determinant factors and their respective relevance through statistical analysis and identifying what mechanisms, unique to OSS development processes, could be applied in domains away from software.

As emphasized above, OSS provides a great amount of online data and we aim to bridge practice with well founded scientific methodology supported by empirical research. In the next sections we describe our methodological approach, related work, our preliminary results and research outlook.

II. Research Methodology

In this section we describe some of the tools provided by complex network methodology and how we seek to apply them to achieve the goals mentioned above. We also describe a set of data that has already been collected and comment on how it will serve to answer the questions considered above.

A. Data Sets

Data on coordination and development processes of OSS projects can easily be found online, concentrated e.g. in software repositories such as SOURCEFORGE and GITHUB. Apart from acquiring the detailed source code history from version control systems like CVS, SUBVERSION or GIT, we have also extracted communication data from publicly accessible forums, mailing-lists and bug tracking systems. Based on this approach, in a preparatory phase of the project we have built a comprehensive database of projects hosted on SOURCEFORGE. We are currently complementing this database by data on source code history and developer collaborations available via the API of GITHUB, which has recently superseded SOURCEFORGE as the largest open source project hosting platform. Apart from these sources, we have further collected data on popular projects like ECLIPSE, which must be crawled separately.

In our project, we lay emphasis on the fact that the purpose of software is likely to give rise do very different architectural patterns, dependency structures and developer-user interactions. In order to study how this affects the development process, we particularly distinguish between projects implementing generic programming frameworks or platforms (like e.g. MONO, ECLIPSE or ASPECTJ) that expose a large part of their inner structure to highly-skilled developers and "end user" software (like e.g. JEDIT or GIMP) which has a much smaller "contact surface" and is mainly being used by non-programmers.

B. Complex Networks Methods in OSS Research

Once data on the evolution of the source code, the interaction between developers as well as the communication between users has been collected, we need to interpret these in terms of networks in order to apply our complex networks approach. An established approach of studying software from the network perspective is to consider functional dependencies between programming abstractions at different scales, like e.g. procedures, types, classes, modules or packages. From a complex networks perspective, data on the evolution of these networks can then be used to check predictions of well-known universal models of network growth [7], to study their correlation with co-change dynamics [8] as well as to model the increase of package incompatibilities [6].

A further domain where taking a network perspective appears promising is in the study of social processes arising in software development. In order to build a simple proxy for a collaboration network, two developers can e.g. be connected to each other if both contribute to the same file (or class, module, etc) within a given time interval (adopted e.g. by [5] in order to identify developers roles). By considering data from OSS forums, mailing-lists and bug tracking systems, we are further building developer and user interaction networks and couple them with both the developer collaboration as well as the software architecture network. Combining these networks and correlating interaction, communication and collaboration events taking place in them promises interesting perspectives for the study of dynamical processes in OSS development.

In recent years, the complex networks community has developed a number of quantitative metrics which capture structural features like e.g. clusters as well as the impact of nodes and clusters on dynamical processes like e.g. information or failure spreading, consensus, opinion formation or synchronization [4]. As an example, the modularity of a software architecture, which is considered a key feature that contributes to the sustainability of large scale projects [9], can be captured by specific network metrics which will be described in section III. In our research, we study how these metrics can contribute to a quantitative understanding of the interdependence of human, social and technical aspects of software engineering.

For the particular research goals outlined in section I, we plan to correlate structural features of dependency and collaboration networks with the underlying programming paradigms and programming languages and study how they impact co-change dynamics and coordination effort. In so far as we are able to identify network structures that exert a beneficial or detrimental effect on the project success, monitoring the evolution of the corresponding quantitative metrics may aid distributed software development and project management. Centrality metrics (like e.g. degree, betweenness or eigenvector centrality) promise to be useful to evaluate the importance of individual developers and users for the project and thus provide interesting perspectives for the study of developer turnover. Following the idea of *socio-*

technical congruence put forth in [10], the co-evolution of communication, collaboration and dependency networks can be studied, thus allowing to shed light on the question how they influence software evolvability, quality and coordination effort. For this, we aim at using the conceptual framework proposed by [11], which provides an interesting approach to the quantitative study of coupled networks.

The larger an OSS project, the less knowledge of the system as a whole individual developers have. From this point of view, OSS development processes can be seen as a complex system. A further interesting aspect of the complex networks perspective on the evolution of network structures is the fact that it allows to relate the micro-level dynamics (e.g. the evolution of dependencies between individual classes and procedures or the interaction between particular developers) with macro-level features of the resulting networks, like e.g. the strengthening or weakening of modularity, the emergence of co-change cascades or fluctuations in coordination events and conflicts. We believe that a solid understanding of this micro-macro link is crucial for the design of development policies and monitoring tools that foster sustainable software engineering.

To summarize, our approach combines quantitative measures from the complex network literature with issues related to software engineering practices. This takes into account the technical aspects of software architecture and also the dynamic processes resulting from social interaction. In the next section we describe the current status of our research.

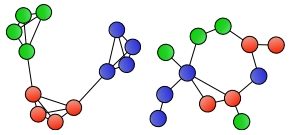
III. Preliminary Results and Publications

This PhD project has recently completed the first year of research. One of our results so far is a modeling framework that allows to describe the growth of tree structures. It augments simpler models for hierarchical networks commonly used in network literature with specific knowledge about software engineering processes (e.g. nodes at different levels in the inheritance hierarchy having different attachment probabilities). We are currently applying it in the analysis of inheritance trees found in object-oriented (OO) programming languages and seek to map it to the evolution of the hierarchical organization of software developers. By studying the position of developers in this hierarchy, we hope to be able to study how developer turnover affects software projects. Part of these results have been presented at the European Conference on Complex Systems 2011 [12], and we are now preparing a submission to a peer-reviewed journal.

A second line of research which we are currently pursuing is the analysis of the role of modularity on software development [13]. Quantitatively, for a given definition of *modules* or *clusters*, a measure of modularity is usually computed in a network framework by

$$Q = \frac{\sum_i^n e_{ii} - \sum_i^n a_i b_i}{1 - \sum_i^n a_i b_i} \qquad (1)$$

where e_{ij} is the fraction of all edges in the network that link nodes in module i to nodes in module j, $a_i = \sum_j^n e_{ij}$, $b_i = \sum_j^n e_{ji}$ (column and row sum respectively) while n is the total number of existing modules. If the network is an undirected graph the matrix defined by $\mathbf{e}$ is symmetric and $a_i = b_i$ [14]. The metric defined by equation (1) measures the fraction of network edges that connect nodes within the same module ($\sum_i^n e_{ii}$) minus the expected value of the same quantity measured from a random network with the same node/module allocation ($\sum_i^n a_i b_i$). If the first is not better than random $Q = 0$ [15]. However, Q would not be defined if all edges are concentrated within a single module because the scaling factor $1 - \sum_i^n a_i b_i = 0$ (no modular structure). In such a case we define $Q = 0$ as well. In general, $Q \in [-1, 1]$, i.e. the more modular the network, the closer Q is to 1. Figure (1) provides two examples of networks and their respective Q scores.



(a) Q=0.8499 (b) Q=0.0545

Figure 1. Network examples where nodes (circles) with the same color are part of the same module. (a) modular network. (b) random connectivity.

In the analysis of software structures, this tool is useful because in many cases module definitions are given by means of programming constructs like classes, namespaces or packages. The Q–metric can thus be used to study how well the cluster structures in the dependency networks corresponds to the modular decomposition of a project in terms of packages, namespaces, etc. In the following analysis, we have applied the Q–metric to the data set described in section II-A. In particular, we have studied the time evolution of the Q–metric of JAVA projects, using class-level dependency networks and a definition of modules based on JAVA packages. In the following, the data are shown for the ECLIPSE IDE, the JEDIT text editor, the AZUREUS torrent client as well as the machine learning library YALE. For the case of JEDIT, the software repository contained the auxiliary and external package structure but not the JEDIT core. For each project, monthly snapshots of the source code were extracted from their respective source code repositories over a maximum period of eight years. Figure (2) shows the time trajectories of each of those projects in a phase space composed of their Q–metric scores as a function of the size of the project (number of classes). We observe that as the project grows, the correspondence of the dependency network to the modular project structure tends to deteriorate.

By studying the release history of each project individually, we were able to verify that development efforts preceding major software releases tend to restore the coherence of modular structures deteriorated previously by the inclusion of new dependencies. This explains for instance

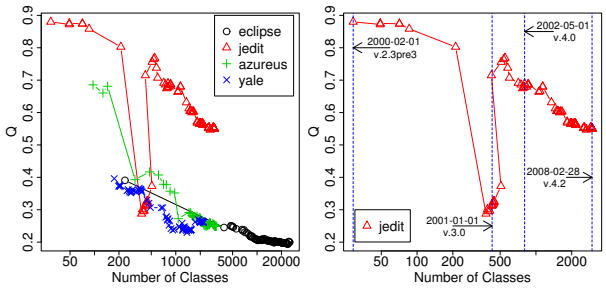


Figure 2. Source code modularity as a function of project size (x axis in log. scale). From left to right, the trajectories in time of four JAVA projects (see text). The trend is to lose modularity as the software project grows in size. (left) all projects. (right) focus on release events of JEDIT.

the drastic fluctuations present in the phase plot of JEDIT (see major releases marked by vertical lines in the right plot of Figure (2)). From the initial version 2.3pre3 to 3.0, a significant deterioration of the Q–metric is observed in a period of less than one year, however the release of version 3.0 did nothing to restore it. According to commonly used version numbering conventions, the major revision number is increased upon a significant redesign of the software. The Q–metric suggests that this is not the case for the release of version 3.0. And indeed, the release notes of version 3.0 justify that the major revision number was increased only because a scripting framework was added to the software. It was only the release of version 4.0 –which according to the release notes re-implemented the document object model as well as the API of the software– which restored the coherence of the cluster structure of dependencies and the modular decomposition of the project to a level at which the architecture could grow further in a sustainable regime which lasted almost six years in our data set.

IV. CONCLUSION AND OUTLOOK

The results shown in section III demonstrate that the Q–metric known from the study of modularity in complex networks is a promising macroscopic approach to study the source code evolution and as described in Section II-B, such a metric could be used as a tool to warn developers about the deterioration of software modularity. It thus provides a simple mapping from local development activity to its respective impact on the system as a whole. However, since we are still in an early stage of our project, these results are necessarily preliminary and further efforts are required to demonstrate the meaningfulness of the Q–metric in terms of sustainability of software development processes. Furthermore –in line with the main goal of our project– we need to expand this approach in order to capture social processes like coordination and communication acts and how they map to the software structures. We hope that this research approach eventually allows us to handle some of the aspects of the co-evolution of the socio-technical structures found in software development.

While we are well aware that the complex networks approach taken by our project has its own specific limitations, we think that it is an important contribution to the field since it allows to integrate the rich results of socio-dynamics with software engineering in a substantiated and quantitative way.

ACKNOWLEDGMENT

We acknowledge the Swiss National Science Foundation for financial support through grant CR12I1_125298 and Ingo Scholtes and Claudio Juan Tessone for valuable comments.

REFERENCES

[1] G. von Krogh and E. von Hippel, "The promise of research on open source software," *Management Science*, vol. 52, no. 7, p. 975, 2006.
[2] C. Gacek and B. Arief, "The many meanings of open source," *IEEE Software*, vol. 21, pp. 34–40, 2004.
[3] M. E. J. Newman, "The structure and function of complex networks," *SIAM review*, pp. 167–256, 2003.
[4] ——, *Networks: an introduction*. Oxford Univ Press, 2010.
[5] M. Pohl and S. Diehl, "What dynamic network metrics can tell us about developer roles," in *ICSE CHASE Proceedings*. ACM, 2008, pp. 81–84.
[6] M. A. Fortuna, J. A. Bonachela, and S. A. Levin, "Evolution of a modular software network," *PNAS*, 2011.
[7] T. Maillart, D. Sornette, S. Spaeth, and G. von Krogh, "Empirical tests of zipfs law mechanism in open source linux distribution," *Phy. Rev. Letters*, vol. 101, 2008.
[8] M. M. Geipel and F. Schweitzer, "The link between dependency and co-change: Empirical evidence," *IEEE Transactions on Software Engineering*, 2011.
[9] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *Software Engineering, IEEE Transactions on*, vol. 11, no. 3, pp. 259–266, 1985.
[10] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the ESEM '08*. ACM, 2008.
[11] S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin, "Catastrophic cascade of failures in interdependent networks," *Nature*, vol. 464, 2010.
[12] C. J. Tessone, F. Schweitzer, and M. S. Zanetti, "Software evolution: From inhomogeneous evolution to coarse-grained dynamics," in *European Conference on Complex Systems, Book of Abstracts*, Viena, 2011, pp. 57–57.
[13] M. S. Zanetti and F. Schweitzer, "A network perspective on software modularity," in *GI LNI Proceedings 200 ARCS Workshops*, 2012.
[14] M. E. J. Newman, "Mixing patterns in networks," *Phy. Review E*, vol. 67, p. 026126, 2003.
[15] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, p. 026113, 2004.