8-6-2011

A Requirements-Based Analysis of Success in Open- Source Software Development Projects

Radu Vlas
Georgia State University, rvlas@cis.gsu.edu

Cristina Vlas
Omnis Prime, cvlas@omnisprime.com

A Requirements-Based Analysis of Success in Open-Source Software Development Projects

Radu Vlas
Georgia State University
rvlas@cis.gsu.edu

Cristina Vlas
Omnis Prime
cvlas@omnisprime.com

ABSTRACT

The literature on open-source requirements is commonly concerned either with the processes associated with these requirements or with very specific requirements-related aspects of open-source development. In this study we bridge between these two approaches by exploring the existing relationships between open-source requirements and few characteristics of open-source projects (software quality and software project success). First we develop a requirements-based taxonomy of open-source projects and we discover patterns linking between this taxonomy and project success. We also propose a classification of requirement types based on their representativeness in open-source projects. This highlights the overall importance of various types of requirements in the context of open-source software development. We also identify exceptions defined as the unusually high frequency of a requirement type and explain them based on the specific domain addressed within the project containing the exception. Finally, we investigate the lifecycle of 16 open-source projects and discover and explain patterns of evolution for a number of requirement types.

Keywords

Open-source, requirements engineering, project success, patterns.

INTRODUCTION

Despite the common belief, open-source software development (OSSD) projects do have requirements. Open-source requirements are, however, mostly informal, lacking a clear, formal organization in well structured documents. Even if sometimes requirements are clearly described, often times they may be only inferred from or suggested in text descriptions found in change requests, bug fixes requests, forums, blogs, email exchanges, and other types of electronic communication. The lack of standardization exhibited by requirements processes and requirements representation in open-source accounts for, among other factors, the difficulty of analyzing this domain and the relative scarcity of research studies exploring this area. Consequently, an in-depth understanding of the nature of OSSD projects and of the evolution of requirements throughout the entire project lifecycle is limited. The ability to better understand open-source requirements and their lifecycle supports an overall better understanding of the nature and evolution of open-source projects. It also provides insights into the mechanisms leading to software product quality and to project success in the open-source domain.

Project success has been extensively explored in the literature on software development and has often been explained through the quality of the software product. The literature on software quality provides several philosophies, models, and approaches accepted by the research community. In this study we support the view according to which requirements quality has a significant and positive impact on software product quality. Literature on project success in the open-source domain highlights few proxies such as the percentage of user recommendations, number of awards, project's ability to meet community needs (percentage of feature and bug requests solved), or number of downloads. While none of these proxies represent a complete solution to defining OSSD project success, we consider a combination of these measures as a way of comparing OSSD projects from a success-oriented perspective. According to Wiegers, a software project's success is mainly determined by characteristics associated with that project's requirements (Wiegers 2003). The success of open-source projects has been acknowledged and studied in a wide variety of industries (Scacchi and Alspaugh 2008; Scacchi, Crowston, Madey and Squire 2009). This research differentiates itself from the existing literature on open-source by exploring the relationship between open-source requirements (e.g. requirements types, distribution, evolution) and open-source software (OSS) product quality and OSSD project success.

Currently, there is a limited understanding of what types of requirements are present in OSSD projects. There is also a lack of studies exploring open source requirements and project lifecycle from a less process-oriented or participant-oriented

perspective and with a higher emphasis on technical aspects (e.g. requirements, design, architecture) related to development. Another knowledge gap in open-source research is represented by the lack of methods and tools for the early estimation of open-source software project success. This study comes to build the foundation for future studies addressing these knowledge gaps.

RELATED RESEARCH

Requirements Classification

Requirements engineering research provides measures that can inform an analysis of development in OSSD. Such measures include requirements types and information on their evolution. Classification provides an overview of the distribution of types of requirements present. Reliability, efficiency, integrity, and usability are common categories. Software quality models, such as McCall (McCall, Richards and Walters 1977), Boehm (Boehm, Brown, Kaspar, Lipow, MaCleod and Merritt 1978), IEEE (1998), and ISO, provide a framework for classifying requirements.

While requirements can be classified many different ways, they are traditionally classified in the research community as either functional (FR) or non-functional (NFR). Some researchers consider this to be too general (Bass, Clements and Kazman 1998). Researchers refer to FRs as goals (or hard goals, or behavioral requirements), and to NFR as soft goals (Mylopoulos, Chung and Yu 1999; vanLamsweerde 2007). FRs specify features of the system to be developed. Therefore, a complete set of FR should comprehensively describe new system's functionality. Non-functional requirements are concerned with two areas: properties that affect the system as a whole (such as usability, portability, maintainability, or flexibility), and quality attributes (such as accuracy, response time, reliability, robustness, or security) (Chung, Nixon, Yu and Mylopoulos 2000; Moreira, Araújo and Brito 2002). Some variations to this classification approach include listing security concerns under FR, adding supportability under NFR, or specifying sub-categories of these two (Grady 1992).

Research on Open-Source Requirements

The open-source domain attracted in the recent years an increasing number of researchers exploring main areas such as software development and social networking analysis, or exploring related areas such as psychology-based perspectives on the open-source participants, the economic impact of open-source, laws and policies affecting the open-source domain, organizational participation in the open-source phenomenon, applicability and use of open-source products in various domains and industries, and natural language processing (NLP) based analysis techniques. In spite of this increased interest in the open-source phenomenon, there still are knowledge gaps that research can address. A number of studies explored OSSD projects and their associated development processes while maintaining a requirements-based perspective (Noll 2008; Scacchi 2002; Scacchi 2009a). Current findings are limited to acknowledging the existence of requirements and their associated processes (elicitation, analysis, specification and modeling, validation, and communication) in open-source projects (Scacchi 2009a). While requirements processes received slightly more attention from a research community apparently focused more on social networking analysis and participant involvement, the end product of OSSD projects, the software product itself and elements that are part of its development lifecycle received a slightly less consistent attention. One important concept that is worth further investigation is represented by open-source requirements and their impact on the quality of open-source software products and on the success of OSSD projects.

Requirements are an important part of any software development project. Requirements lifecycle analysis is a challenging task mainly because requirements evolve continuously along with the evolution of the software development project. At the very bottom of this continuous change lie social factors (stakeholders' different views) and development factors (production constraints, usage experience, or feedback received) (Anderson and Felici 2002). In spite of a consistent body of research in the area, requirements evolution models and analysis methodologies are still providing an incomplete solution to the problem. There are two challenges defining this situation: first, the large quantity of longitudinal data that must be collected and analyzed and second, the lack of methodologies to provide real-time support for analyzing requirements evolution (Jarke and Paulk 1994; van Lamsweerde 2000). If these circumstances can be overcome, understanding requirements evolution can be a very useful tool for identifying rare environmental events or for providing strategies to deal with environmental changes (Lutz and Mikulski 2003).

In the process of better understanding the OSS product development, researchers focused on defining and analyzing requirements (Noll 2008; Scacchi 2002). Research studies exploring development processes in open-source concluded that there are no formal processes similar to the ones closed-source development employs in requirements engineering. However, Scacchi identified informal processes similar to requirements elicitation, analysis, specification, validation, and management (Scacchi 2002, 2006, 2009b). These processes have a significant social component and define the general characteristics of

requirements lifecycle in open-source. In many cases, open-source software projects adopt a post-hoc approach in which requirements take shape after their corresponding implementation in the developed product is realized.

Open-source requirements are commonly part of informal communication, rather than the result of classical formal modeling (Scacchi 2002). They emerge in open-source through a dynamic social process of communication among open-source participants. Requirements are developed in open-source through a wide variety of web-based communications that were defined as software informalisms (Scacchi 2002). The informal component of open-source requirements is confirmed by Lintula who acknowledges the importance of discussion forums as a means of reaching common understanding and acceptance on requirements (Lintula, Koponen and Hotti 2006). Given the informal nature of open-source requirements, a manual analysis is time-consuming and error-prone, especially in the case of large projects. Vlas and Robinson recently acknowledged this shortcoming and proposed a model of requirements composition to provide a consistent perspective on requirements and their components. They also propose an associated method for automated discovery and classification of natural language (NL) requirements. Their classification approach uses McCall's software quality model for defining a set of 23 quality-oriented classification types (Vlas and Robinson 2011).

Software Product Quality and Software Development Project Success

Success is an important attribute associated with any software development project. While this concept seems simple and intuitive, there is little agreement as to what constitutes software quality or project success (Pinto and Slevin 1987). Traditionally, projects were perceived as successful when they met time, budget, and performance goals. Obviously "success" is more than meeting budget and deadlines. TAM (Davis 1989; Venkatesh, Morris, Davis and Davis 2003) posits that perceived usefulness and perceived ease of use determine an individual's decision to use a system, which in turn determines that project's success. Taking into consideration that different groups of stakeholders have different views on success, other dimensions have been defined as determinants of a project's success: project efficiency, impact on customer, business success, and preparing for the future (Shenhar, Dvir and Levy 1997). DeLone and McLean identified six dimensions of success: "systems quality" which measures technical success, "information quality" which measures semantic success, and "use, user satisfaction, individual impacts" and "organizational impacts" which measure effectiveness success (DeLone and McLean 2003). The quality of information, system, and service leads to higher user satisfaction, which leads to user's intention to use the product, and certain net benefits occur.

In the last years, researchers underlined the importance of requirements as a determinant of project success: "requirements are essential for creating successful software because they let users and developers agree on what features will be delivered in new systems" (Wiegers 2003). Early user involvement has been related to higher requirements quality. Empirical studies confirmed this and showed that involving users and customers as sources of information is related to project success (Kujala, Kauppinen, Lehtola and Kojo 2005). The relationship between requirements quality and project success is explored by Hooks and Farry (Hooks and Farry 2001). They show that the two variables are positively correlated. Dvir summarizes these findings by stating that user involvement in the development of requirements is positively and significantly correlated with the overall success of a project (Dvir 2003, 2005). Therefore, requirements and their evolution represent two of the essential attributes of software development projects and require a special attention in studies on project success.

METHODOLOGY

Linking attributes of requirements to software quality and project success can be done by considering quality to be a main independent variable in a model predicting success and by extending the assumption of quality from the requirement level to the software product level and subsequently to the project level (Crowston, Annabi and Howison 2003; DeLone and McLean 1992). Wiegers' and Dvir's findings that software project success is determined by that project's requirements directly support this perspective (Dvir 2005; Wiegers 2003). In this exploratory research, we explore the concept of requirements quality by using Vlas and Robinson's method of classifying open source requirements through a mapping to a generally accepted software quality model, such as McCall's (McCall et al. 1977; Vlas and Robinson 2011). Therefore, McCall's 23 software quality criteria determine the taxonomy of 23 types used to classify open-source requirements (Figure 3). The goal of this research is to adopt a design-oriented perspective while exploring characteristics of OSSD projects. This approach gives us the opportunity to discover the impact of open-source requirements on three dimensions of success central to the IS success model proposed by De Lone and McLean. Specifically, we extend the assumption of quality from the open-source requirements level to the system level and we consider patches solved and feature requests met as indicators of system quality. We also explore the indirect impact open-source requirements exhibit on project success (via characteristics of system use and indicators of user satisfaction). We use product downloads as an indicator of system use, and percentage of user recommendations and number of awards as indicators of user satisfaction.

In this study we use the open-source requirements discovery and classification method and tool (RCNL) proposed by Vlas and Robinson (Vlas and Robinson 2011). We perform natural language processing (NLP) analysis of open-source projects only to the extent that allows us to replicate the study by Vlas and Robinson. For classifying discovered requirements we use the enhanced set of classification rules (McCall+) provided by RCNL. Through the mapping of requirements to the 23 software quality criteria proposed by McCall, we extend the assumption of quality to the requirement level and, consequently, to the software system level.

Since our study is an exploratory study of OSSD projects' lifecycle, we are concerned with the interpretation of requirements-related information we obtain from using the RCNL tool. We place this information in the context defined by other measures related to OSSD projects and we identify a set of findings through reasoning and logical inference. For the validation of these findings, we extract and analyze additional information on OSSD projects. The findings are validated when they are confirmed by the project characteristics identified in the additional information collected.

Data Collection and Analysis

We use same source of data as the original study by Vlas and Robinson. Therefore, for our dataset we select the same set of 16 SourceForge projects (Figure 6) and we extract data from the same February 2010 data dump. As explained by Vlas and Robinson, the dataset includes projects that are considered to be active. The projects were selected based on the number of participants, the number of downloads, and the number of feature requests. Therefore, we acknowledge the possibility that our dataset includes more successful than unsuccessful projects as one of the main limitations of our dataset. Consequently, we also acknowledge the limited external validity of our findings. While unsuccessful projects might be under-represented in our dataset, the selection criteria we use do not stop less successful projects from being included in the dataset. Our data collection uses the online access offered by Notre Dame University to SourceForge data through the SourceForge Research Data Archive (SRDA) (Gao, Antwerp, Christley and Madey 2007). Data collected is organized in 16 text files, one for each project. Each of the project files contains all feature requests postings associated with that specific project, and listed in chronological order. The timestamp for each posting is also included in our data files.

The analysis of OSSD projects lifecycle requires a time-based analysis of available data. We use the included timestamps to determine the duration of each project and we split up the project files into 6 months long data windows. The length of the last data window in each project is between 3 months and 9 months in order to include all feature requests postings available. For each data window created, we process feature requests with the RCNL tool in order to discover and classify requirements. The results associated with last data window in each project might be slightly skewed upward or downward as a result of the different duration of the data window. Our analysis of results takes into account this aspect.

The requirements-based analysis of OSSD projects lifecycle includes within project analysis and cross-project analysis. We explore the evolution of the number of requirements as a factor who shapes a project's lifecycle. We start with an analysis of the evolution of the overall number of requirements. This helps us identify main patterns of project evolution. Next, we analyze the evolution of individual types of requirements throughout the duration of a project and we identify patterns of requirements types' evolution. We also identify patterns of evolution for groups of requirements types. Next, we correlate requirements results with project-specific data. Here we compare patterns of evolution across projects and we discover the relationships between project type and type of evolutionary pattern. Finally, we place all patterns of evolution discovered in a broader project-related context in order to validate them. This more general context is constructed from additional project information such as project type and description, number of positive recommendations and awards, release dates and types, percentage of feature requests solved, and number of downloads. We collect this information from SourceForge.

RESULTS

The set of requirements-based analyses we consider generate a number of interesting results. First we plot the overall number of requirements discovered per data window for the entire duration of the 16 projects. The resulting graphs indicate the existence of taxonomy of open-source project lifecycle types. We identify 3 main types of open-source project lifecycles: bell-shaped, half bell-shaped, and unstable. We further classify the unstable type of projects as either "double-spiked" or "full unstable" projects. Figure 1 presents these project types. In our dataset of 16 open-source projects, 4 belong to *type a* (bell-shaped), 3 belong to *type b* (half bell-shaped), 6 belong to *type c* (unstable: double-spiked), and 3 belong to *type d* (unstable: full unstable).

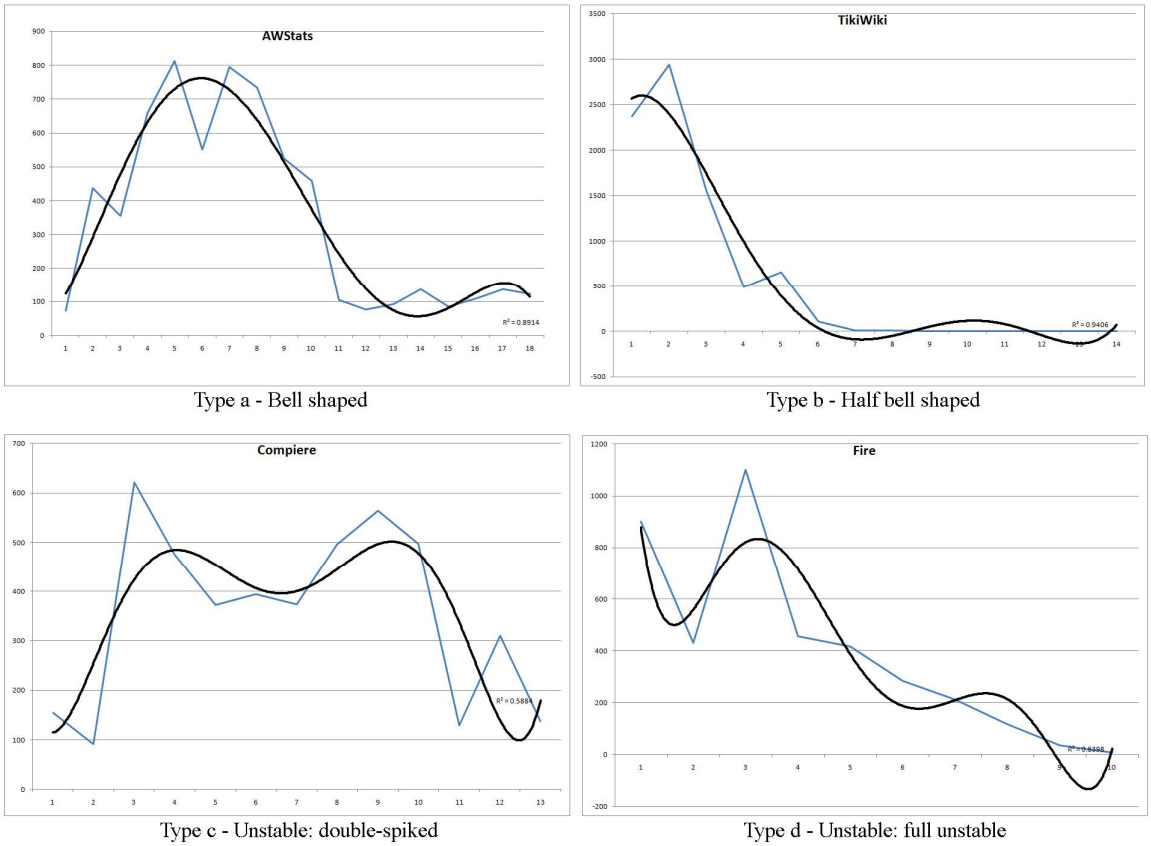

Figure 1. The taxonomy of open-source project types.

The apparently unusual and unpredictable variations in the number of requirements per data window for a project can be explained by various events in the lifecycle of the projects. In order to explore this relationship, we extract information on the number and date of all types of software releases and on the number of new features implemented in these releases. When plotting the number of new features released, we find that open-source projects' lifecycles are continuously influenced by and react to the behaviors of communities surrounding them. Figure 2 presents the case of the KeePass project.

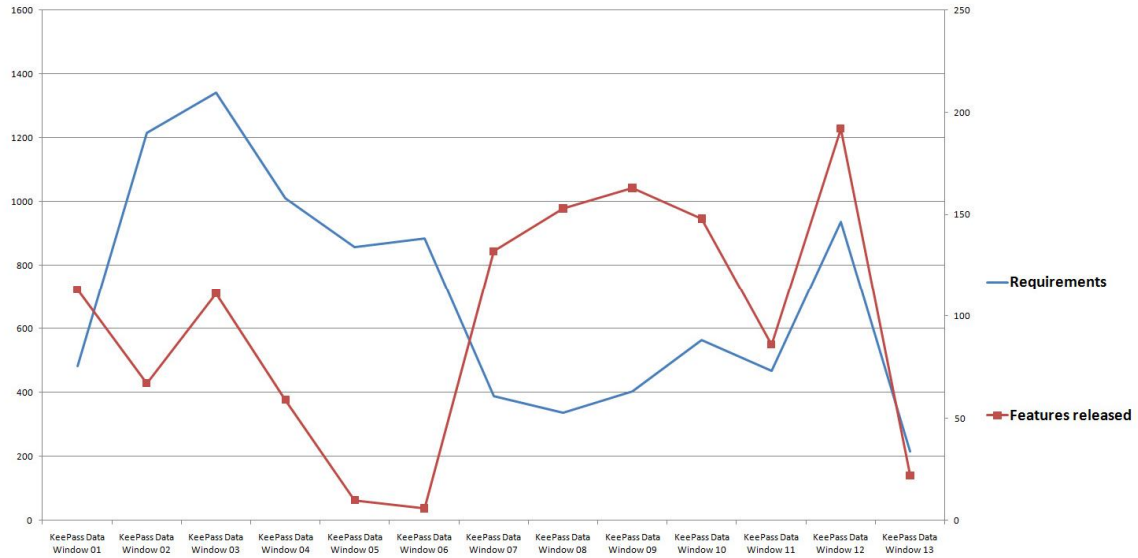

Figure 2. KeePass project – Evolution of features released and requirements.

During first 3 years of the project (first 6 data windows), the set of desired features mentioned in project community's discussions (requirements identified in feature requests – blue line) is not matched by the number of features included in the project releases (red line). This suggests 2 effects: an accumulation of desired features, and a general discontent of the project community, which reflects in a decline of the number of feature requests postings. Beginning of the 4th year marks a significant change as a large number of features is released. This seems to indicate an attempt to implement those features corresponding to the accumulation of requests from the first 3 years of the project. This also reactivates the interest of the community and fuels discussions on the feature requests posting board. Therefore, the number of requirements discovered seems to be on a slightly positive trend. It can be assumed that the implementation of large numbers of features in data windows 7 through 10 allowed the KeePass team to catch up with the requests for features. This explains the matching trends exhibited by the number of requirements discussed in feature request postings and the number of features released. KeePass' lifecycle highlights a three-step evolutionary pattern also exhibited by other projects in our dataset. In step 1, the project does not release enough features to cover for the amount of discussion in the project community. Step 2 indicates a move towards project maturity and is defined by a continuous effort to implement the accumulated number of features requested through frequent major releases. In Step 3, the project is mature and capable of matching current requests for new features.

The taxonomy of requirements types includes 23 criteria which we classify as "high representation (HR)," "medium representation (MR)," or "low representation (LR)" type based on their percentage out of the total number of requirements. We find that in our dataset 4 criteria consistently rank as HR, 7 criteria consistently rank as MR, and 12 criteria consistently rank as LR. These criteria and their type are presented in Figure 3. It is important to note here that project type has an impact on the type of requirements that are normally classified as LR or MR within a project. For instance, KeePass is a password manager system and, therefore, access control (C8) and access audit (C9) are vital to its success. PhpMyAdmin is a database related project and storage efficiency is one of its main concerns. These exceptions are highlighted in Figure 3.

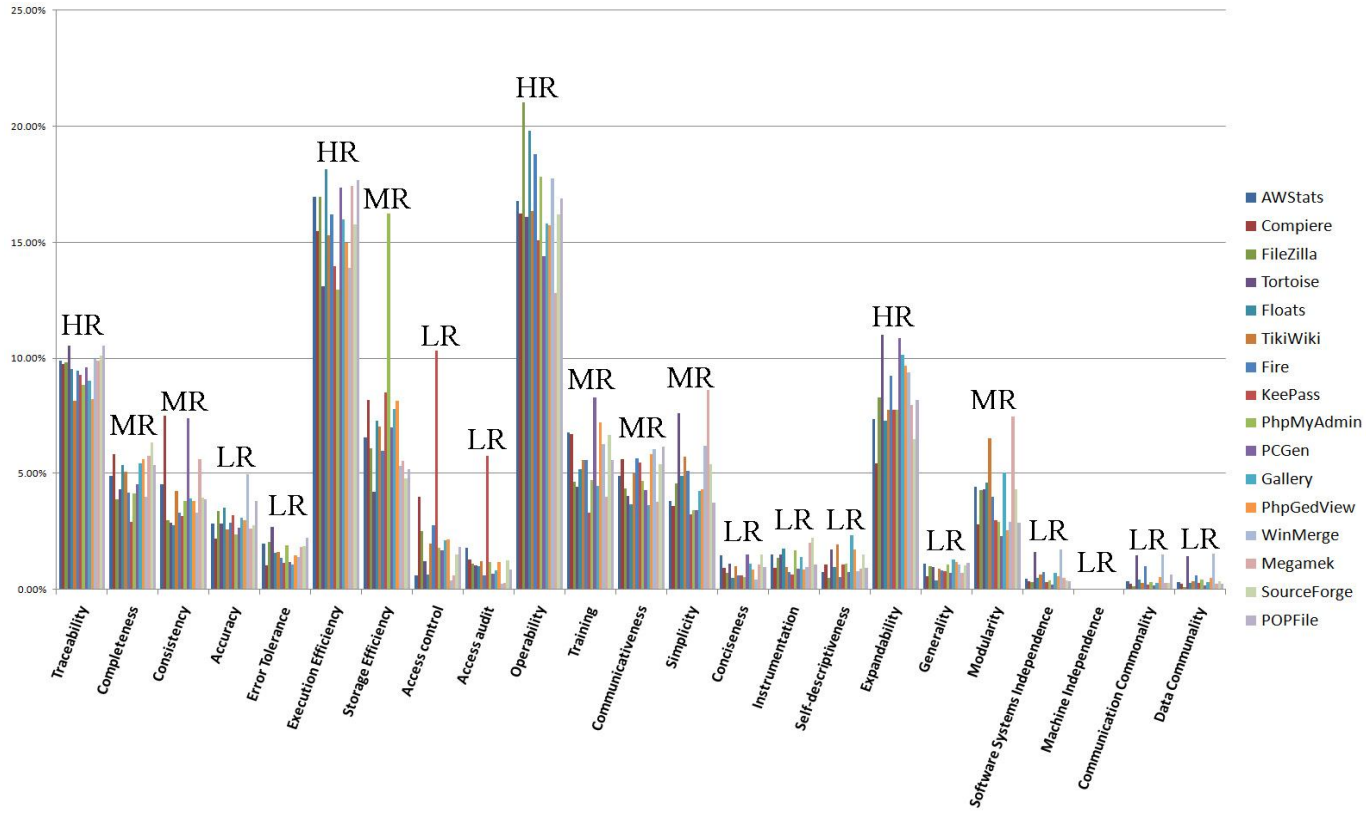

Figure 3. The taxonomy of requirements across open-source projects.

The inspection of distinct requirements types throughout a project's lifecycle identified few interesting patterns of evolution. First, we notice a decrease in project activity (volume and number of postings in feature requests forums) immediately after a spike in traceability (C1). The need to trace back requirements and other various characteristics of software during testing is known and generally accepted. Thus, increases of traceability indicate an increase in testing activities. This is normally associated with the preparing of new major releases of software. With every major release, a large portion of the existing

requests for features is addressed and the volume of discussion on feature requests forums is expected to decrease. We also notice that traceability tends to become increasingly important towards the end of a project, sometimes surpassing other leading types of requirements. This is justified by the increase of testing activities in the last stages of software projects. For example, in the PCGen project, expandability (C17) is one of the leading types in the first part, but traceability (C1) becomes better represented than expandability in the second part. Similarly, access control (C8) is a leading type during first part of KeePass project, while storage efficiency (C7) is a leading type during first part of Compiere project. In the last part, traceability (C1) surpasses access control (C8) and storage efficiency (C7), respectively.

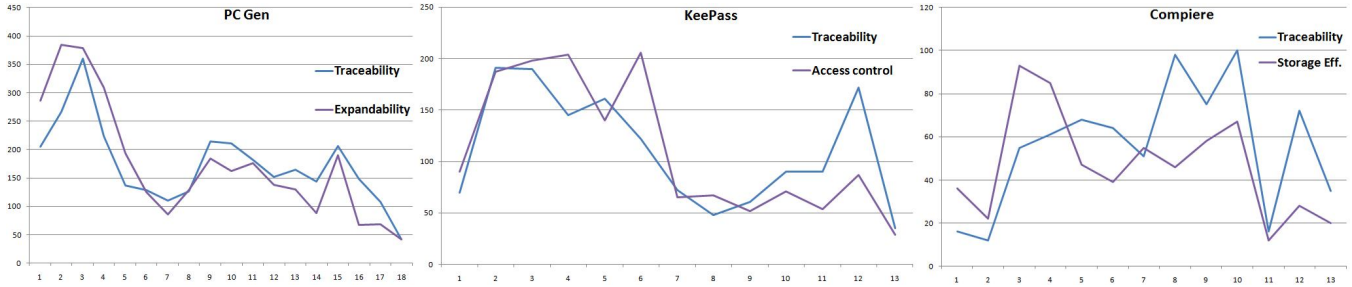

Figure 4. Evolution of traceability and other leading requirements types for PCGen, KeePass and Compiere projects.

Tortoise exhibits another interesting pattern of evolution. Initially, expandability is the focus of development and this can be justified by the efforts necessary to build the system. In the middle of project's evolution, traceability seems to become more important, which indicates an attempt to test and release major functionality. This explains the general downward trend exhibited by Tortoise for a period of 3 years in the middle of its evolution. Last part of the project is characterized by a renewed interest in expandability, which seems to indicate an attempt to revamp the project by building new functionality into it, while the overall interest in this project is very low. This evolution is presented in Figure 5.

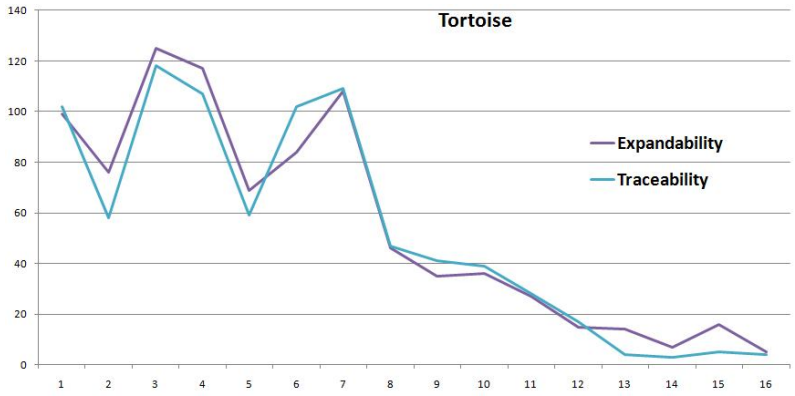

Figure 5. Evolution of traceability and expandability in the Tortoise project.

Figure 6 presents a summary of main project-level information we collected. This information includes few rough proxies of success. We sorted our set of 16 projects by the average number of weekly downloads reported by SourceForge. While none of these indicators provides a complete image of success, they complement each other for distinguishing more successful projects from the less successful ones. We conclude that Filezilla, KeePass, phpMyAdmin, and MegaMek, are among the most successful projects while Fire, and TikiWiki, are among the least successful projects. The four projects we indicate as successful fall under type a (bell-shaped) or type c (unstable: double-spiked). In contrast, the two projects we identified as unsuccessful are of type d (unstable: full-unstable) and type b (half bell-shaped). The 2 unsuccessful projects do not exhibit an unusual behavior with regards to the other requirements-based patterns. Two of the successful projects exhibit an unusually high percentage of requirements types specific to their areas (KeePass – accessibility, phpMyAdmin – data storage and administration). MegaMek shows high percentages of modularity and simplicity requirements. This is explained by the need for a simpler, modular architecture that games normally require in order to be able to offer expandability and flexibility over time.

Project Name	Official Project Description from Project's Webpage	Recommended by	Awards	Feature Reqs Solved	Patches solved	Weekly downloads	Reqs discovered
FileZilla	FileZilla is a cross-platform graphical FTP, FTPS and SFTP client a lot of features, supporting Windows, Linux, Mac OS X and more. FileZilla Server is a reliable FTP server for Windows.	89%	2	x	x	714553	9231
KeePass	KeePass Password Safe is a free, open source, light-weight and easy-to-use password manager for Windows. You can store your passwords in a highly-encrypted database, which is locked with one master password or key file.	84%	0	79%	99%	114068	7489
PhpMyAdmin	MySQL over the Web. Currently it can create and drop databases, create/drop/alter tables, delete/edit/add fields, execute any SQL statement, manage keys on fields.	84%	7	77%	92%	57516	6623
WinMerge	WinMerge is a Windows tool for visual difference display and merging, for both files and directories. Unicode support. Flexible syntax coloring editor. Windows Shell integration. Regexp filtering.	93%	0	61%	97%	31148	4517
Gallery	A slick, intuitive web based photo gallery. Gallery is easy to install, configure and use. Gallery photo management includes automatic thumbnails, resizing, rotation, and more. Authenticated users and privileged albums make this great for communities	79%	1	73%	76%	15214	12112
Tortoise	TortoiseCVS is an extension for Microsoft Windows Explorer that makes using CVS fun and easy. Features include: coloured icons, tight integration with SSH, and context-menu interactivity.	92%	1	61%	95%	5163	5615
AWStats	AWStats is a free powerful and featureful server logfile analyzer that shows you all your Web/Mail/FTP statistics including visits, unique visitors, pages, hits, rush hours, os, browsers, search engines, keywords, robots visits, broken links etc	83%	1	31%	63%	4169	6260
Float's	Phonebook Manager, Organizer, Fun and much more; whatever you want it to be, it is whatever a mobile phone should have :-) (Currently based on Sony Ericsson features set)	85%	0	28%	100%	2073	4124
PCGen	PCGen is a free open source RPG character generator (d20 systems). All datafiles are ASCII so they can be modified by users for their own campaigns.	92%	0	x	x	1829	15717
TikiWiki	Powerful multilingual Wiki/CMS/Groupware to build & manage your Wiki, Files/Image Galleries, CMS, Blog, Tracker/Forms, Forums, Directory, Polls, Surveys, Quizzes, Newsletters, Calendars, FAQs, Spreadsheets, Maps, Workflow, etc. - That all is Tiki!	87%	1	23%	61%	1550	8135
MegaMek	MegaMek is a networked Java clone of BattleTech, a turn-based sci-fi boardgame for 2+ players. Fight using giant robots, tanks, and/or infantry on a hex-based map.	96%	1	82%	98%	922	6635
PhpGedView	PhpGedView is a revolutionary genealogy program which allows you to view and edit your genealogy on your website. It has full privacy functions, can import from GEDCOM files, and supports multimedia. It also simplifies family collaboration.	93%	0	49%	89%	766	6257
Compiere ERP	Compiere ERP+CRM is the leading open source ERP solution for Distribution, Retail, Manufacturing and Service industries. Compiere automates accounting, supply chain, inventory and sales orders. Compiere ERP is distributed under GPL V2 by CompiereInc.	48%	1	53%	97%	718	4615
Fire	Fire is a multi-protocol instant messenger client for Mac OS X based on freely available libraries for each service. Currently Fire handles AOL Instant Messenger, ICQ, MSN Messenger, Jabber, limited IRC, Yahoo!, and Apple Bonjour communications.	62%	0	80%	0%	269	3964
SourceForge	Official Support and Documentation for SourceForge.net, provided by the SourceForge.net Service Operations Group (SOG).	82%	0	x	x	216	19832
PopFile	POPFile is an email classification tool with a Naive Bayes classifier, POP3, SMTP, NNTP proxies and IMAP filter and a web interface. It runs on most platforms and with most email clients.	78%	1	88%	97%	32	8596

Figure 6. Project-level information.

CONCLUSIONS

This study presents the findings of a requirements-based analysis of a dataset of 16 open-source projects. These findings include patterns of evolution for OSSD projects and for specific types of requirements identified in the study. The patterns are validated through logical reasoning based on additional information we extract from SourceForge. Final findings determine an initial set of characteristics separating more successful projects from the less successful ones. This study explores an area of research little addressed by open-source researchers and represents an initial step towards a better understanding of the relationships between the evolution of design and architecture elements and the success of OSSD projects. Open-source project leaders can benefit from the results of this study by having a means of determining if the project falls outside of the set of patterns commonly exhibited by successful projects. Specifically, open-source project champions can attempt to compare their project's characteristics to the set of patterns identified in this study and associated with more successful projects. If a match is not found, then project leaders can consider taking corrective actions while using the desired patterns of success to guide their efforts.

We acknowledge a number of limitations. First, the discovery of requirements in postings of feature requests is not an error-proof process. Similarly, the RCNL-based classification of requirements leaves some of them not classified. Despite these, we consider the performance measures of the RCNL tool to justify its use in this study. Our dataset consists of only 16 open-source projects and these are selected based on three criteria, one of which indicates success. We call on researchers to replicate this analysis on datasets with a larger representation of failed projects, and on projects from other domains. Other directions for future research include the development of a different taxonomy of requirements types, and a more in-depth (ethnographic) study of open-source projects as a way of identifying events determining project lifecycle changes.

REFERENCES

1. (1998). "IEEE standard for a software quality metrics methodology." *IEEE Std 1061-1998*.

2. Anderson, S. and M. Felici (2002). Quantitative Aspects of Requirements Evolution. *Proceedings of the 26th Annual International Computer Software Conference, COMPSAC 2002*, Oxford, England, IEEE Computer Society Press.

3. Bass, L., et al. (1998). Software Architecture in Practice. Reading, MA, Addison Wesley.

4. Boehm, B. W., et al. (1978). Characteristics of Software Quality. New York, North-Holland.

5. Chung, L., et al. (2000). Non-functional Requirements in Software Engineering, Springer.

6. Crowston, K., et al. (2003). Defining Open Source Software Project Success. *Proceedings of the 24th International Conference on Information Systems*.

7. Davis, F. D. (1989). "Perceived usefulness, perceived ease of use, and user acceptance of information technology." *MIS Quarterly* **13**(3): 319-339.

8. DeLone, W. H. and E. R. McLean (1992). "Information Systems Success: The Quest for the Dependent Variable." *Information Systems Research* **3**(1).

9. DeLone, W. H. and E. R. McLean (2003). "The DeLone and McLean Model of Information Success: A Ten-Year Update." *Journal of Management Information Systems* **19**(4): 9-30.

10. Dvir, D. (2003). "An empirical analysis of the relationship between project planning and project success." *International Journal of Project Management* **21**(2): 89-95.

11. Dvir, D. (2005). "Transferring projects to their final users: The effect of planning and preparations for commissioning on project success." *International Journal of Project Management* **23**(4): 257-265.

12. Gao, Y., et al. (2007). A Research Collaboratory for Open Source Software Research. *Proceedings of the 29th International Conference on Software Enginering + Workshops (ICSE-ICSE Workshops 2007), International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS 2007)*, Minneapolis, MN.

13. Grady, R. (1992). Practical Software Metrics for Project Management and Process Improvement. Englewood Cliffs, NJ, Prentice Hall.

14. Hooks, I. F. and K. A. Farry (2001). Customer-Centered Products: Creating Successful Products Through Smart Requirements Management, Amacom.

15. Jarke, M. and K. Paulk (1994). "Requirements Engineering in 2001: (virtually) managing a changing reality." *Software Engineering Journal*: 257-266.

16. Kujala, S., et al. (2005). The role of user involvement in requirements quality and project success. *Proceeding of the 13th IEEE International Conference on Requirements Engineering*, Helsinki University of Technology, Finland, Software Business & Engineering Institute.

17. Lintula, H., et al. (2006). Exploring the Maintenance Process through the Defect Management in the Open Source Projects - Four Case Studies. *Proceedings of the International Conference on Software Engineering Advances (ICSEA'06)*, Como, Italy.

18. Lutz, R. R. and I. C. Mikulski (2003). "Operational Anomalies as a Cause of Safety-Critical Requirements Evolution." *The Journal of Systems and Software* **65**(2): 155-161.

19. McCall, J. A., et al. (1977). Factors in Software Quality, NTIS.

20. Moreira, A., et al. (2002). Crosscutting Quality Attributes for Requirements Engineering. *Proceedings of the Software Engineering and Knowledge Engineering Conference (SEKE)*, Ischia, Italy.

21. Mylopoulos, J., et al. (1999). "From Object-Oriented to Goal-Oriented Requirements Analysis." *Communications of the ACM* **42**(1): 31-37.

22. Noll, J. (2008). Requirements Acquisition in Open Source Development: Firefox 2.0. Open Source Development, Communities and Quality, IFIP International Federation for Information Processing**:** 69-79.

23. Pinto, J. K. and D. P. Slevin (1987). "Critical factors in successful project implementation." *IEEE Transactions Engineering Management* **EM-34**(1): 22–27.

24. Scacchi, W. (2002). "Understanding the Requirements for Developing Open Source Software Systems." *IEEE Proceedings - Software* **149**(1): 24-39.

25. Scacchi, W. (2006). Understanding Free/Open Source Software Evolution. Software Evolution and Feedback: Theory and Practice. J. F. R. a. D. P. e. N.H. Madhavji. New York, John Wiley and Sons, Inc.**:** 181-206.

26. Scacchi, W. (2009a). Understanding Requirements for Open Source Software. Design Requirements Engineering – A Multi-disciplinary perspective for the next decade. K. Lyytinen, P. Loucopoulos, J. Mylopoulos and W. Robinson, Springer-Verlag.

27. Scacchi, W. (2009b). Understanding Requirements for Open Source Software. Design Requirements Engineering: A Ten-Year Perspective. P. L. K. Lyytinen, J. Mylopoulos, and W. Robinson (eds.). Heidelberg, Germany, Springer-Verlag**:** 464-494.

28. Scacchi, W. and T. Alspaugh (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense. The 5th Annual Acquisition Research Symposium.

29. Scacchi, W., et al. (2009). Envisioning National and International Research on the Multidisciplinary Empirical Science of Free/Open Source Software.

30. Shenhar, A. J., et al. (1997). "Mapping the dimensions of project success." *Project Management Journal* **28**(2): 5-13.

31. van Lamsweerde, A. (2000). Requirements Engineering in the Year 00: A Research Perspective. *Proceedings of the 2000 International Conference on Software Engineering, ICSE 2000*, Limerick, Ireland.

32. vanLamsweerde, A. (2007). Goal-Orientation in Requirements Engineering. Requirements Engineering - From System Goals to UML Models to Software Specifications, Wiley.

33. Venkatesh, V., et al. (2003). "User acceptance of information technology: Toward a unified view." *MIS Quarterly* **27**(3): 425-478.

34. Vlas, R. and W. Robinson (2011). A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. *Proceedings of the 44th Hawaii International Conference on Systems Science*, Kauai, HI.

35. Wiegers, K. E. (2003). Software Engineering. Redmont, Washington, Microsoft Press.