Evolution of Open Source Software: A Study of the Samba Project

Mae Lyn LEE and Joseph DAVIS

School of Information Technologies, The University of Sydney, Australia

ABSTRACT

Open Source Software (OSS) development model has attracted considerable attention in recent years, primarily because it offers a non-proprietary and socially beneficial model of software development backed by a dedicated community of developers and users who share and expand their knowledge and expertise. This research investigates the evolution of open source software using a case study of the Samba project Through the application of both qualitative and quantitative techniques, Samba's software development and evolution over a seven-year period are tracked and assessed This assessment and the findings of similar, previously reported studies lead us to propose a general framework for the evolvabiltity and the key drivers of open source software evolution.

Key-words: Open source, Software development, Software evolution, Open source community.

RÉSUMÉ

Le modèle du développement du logiciel Libre a attiré une attention considérable ces dernières années, principalement car il offre un modèle non-propriétaire et socialement bénéfique supporté par une communauté de développeurs et d'utilisateurs dédiés qui partagent et étendent leurs connaissances et expertise. Cette recherche investigue l'évolution du logiciel Libre à travers l'étude de cas du projet Samba. Par des techniques quantitatives et qualitatives, nous décrivons et évaluons l'évolution du développement de ce logiciel. Cette évaluation conduit à proposer un cadre d'analyse général de l'évolutivité et des facteurs clés de l'évolution des logiciels libres

Mots-clés : Logiciel Libre, Développement logiciel, Évolution du logiciel, Communauté du Libre.

I. INTRODUCTION

Open source software (OSS) development has emerged as a progressive and successful software development practice in recent years. Unlike traditional software development, OSS development lacks stringent control over the development process, relying mainly on a pool of volunteer developers for their progress. The open source community conducts and coordinates its activities without strict guidelines and yet it has had success in developing robust and efficient software.

There are thousands of active open source software projects at any time at various stages of development. The successes of key OSS projects such as the Linux operating system and Apache web server have served to draw scholarly focus on the OSS development model (Cubranic, 1999). Each project is undertaken by dozens or hundreds of developers who are geographically distributed all over the world and coordinate their activities through the internet. They bring a variety of expertise and skills and contribute at varying levels of intensity. Their primary motivation appears to be peer recognition and personal satisfaction (Moody, 2001). A number of authors have pointed out that the main characteristics and modes of functioning of the virtual organisation that is behind each of these projects (Browne, 1998; Markus, Manville, and Agres, 2000). The latter studied the virtual organisations behind the OSS projects and suggested that the multiple interacting governance mechanisms, rules and institutions, monitoring and sanctions, and the reputation systems of OSS projects offer a viable, alternative model for traditional commercial and other organisations.

The perception of relative success of OSS has triggered the question of whether there are aspects of the open source software development process that can provide an explanation. More importantly, how does the OSS development find structure in a seemingly chaotic and uncoordinated environment? Consequently, the unorthodox software development techniques developed by the open source community have attracted interest. There have been previous studies on open source evolution with Linux and the Apache, two of the most popular and successful open source software. We extend this research through a case study of the Samba open source project. Samba software is one of the fastest growing OSS on the market. Its popularity is attributed to Samba providing unique services across different platforms. Samba is a suite of Unix applications with the ability to mediate between Unix and Windows systems, using the Server Message Block (SMB) protocol, also referred to as CIFS (Common Internet File System) protocol. Samba has contributed significantly ameliorating the inter-operability problems in contemporary information systems environments.

This research examines Samba project to assess its growth and evolution over a seven-year period across five releases. By employing both quantitative and qualitative measurement techniques, Samba's development and growth can be traced and compared against guidelines of software evolution proposed by Lehman et al. (2001).

Samba's development community was also studied to explore the relationship between Samba's growth and the degree to which the community drives the evolution process.

I.1. Overview of Open Source Software

Open Source software is software that is freely available for public use, including the source code and its modification and redistribution. In the 1970s, Richard M. Stallman, of the Massachusetts Institute of Technology (MIT) Artificial Intelligence (AI) Research Group created the Free Software Foundation (FSF) to counteract the growing secrecy of software source codes in the proprietary software world. Stallman viewed the commercialisation of software as an impediment to the development of an open learning environment needed for software development. Stallman argued that free software should provide users with the following freedoms (Gacek et al., 2001, Feller and Fitzgerald, 2000):

- The program and the source code must be freely available to the public;
- The software can be reproduced and must be redistributed, either gratis or for a fee;
- The software can be studied and must be modifiable;
- Derivative works of this software must be permitted and available for distributions.

These conditions for free software are reflected in the GNU General Public License (GPL) which is more stringent than range of open source licenses currently in use.

The success of Open Source software has been attributed to many factors including the following identified by Schmidt (2001), Wang (2001), and Yamauchi et al. (2000):

- The reduction in software development and acquisition cost;
- Short feedback loops;
- Effective leverage of user community expertise and computing resource;
- Greater opportunity for analysis and validation.

Open Source software can maintain a low software development and acquisition cost base given that it is not subject to development fees, licensing fees or distribution costs. OSS projects utilise low-cost communication media such as the Internet for the distribution and development of the software. This also facilitates convenient access to the open source project itself, allowing external users and developers to provide informative feedback and criticisms on the software structure and potentially problematic bugs and errors. Eric Raymond attributes the quick discovery of software defects in OSS to "*many eyeballs looking for problems*" (Raymond, 2001).

The ability to draw resources from a pool of high calibre software professionals has seemingly contributed to the quality of OSS products and created a community spirit. It has encouraged active information sharing and promotes innovation in software development. Open source projects are evaluated not only in terms of program

execution; its source code is scrutinized by a large and openly active community of professional developers and users. Traditional hierarchical and closed software development and quality assurance processes rarely achieve the benefits of such extensive communication and feedback.

Along with the benefits of OSS development, there are important challenges. Schmidt and Porter (2001) have outlined some of the major challenges:

- OSS is faced with long-term maintenance issues, including the cost of evolution and providing quality assurance. The cost factor must be limited to ensure OSS remains publicly available, including its program and source codes;
- OSS frequently releases beta versions of the software which can create problems for end-users who rely on stable software releases, and would prefer less frequent releases of patches and bug fixing. These frequent beta releases result from short feedback loops in which bugs and previous errors are immediately detected and corrected;
- OSS is faced with ensuring coherence of system-wide properties, including enforcing standardized common APIs, consistent semantics and architectural integrity. OSS developer communities are usually decentralized and hence project and community coordination are required to ensure that the system components can be integrated to operate efficiently.

These challenges are currently tackled by the OSS community with assistance from open access to source codes, ubiquitous web access for sharing and feedback of existing and new software versions, and a growing technologically-sophisticated, global community of users and developers alike.

I.2. Open Source Development Communities

Several authors have argued that hierarchical and relatively closed systems do not give the developers sufficient control over their own knowledge and tend to place barriers between creation and integration of software. As a result, innovations tend to take place outside these systems and they work with information which is very often chronically out of date and which reflects an outsider's view of work (Brown and Duguid, 2000, Scacchi, 2001).

OSS communities exhibit certain common characteristics. Scacchi (2002) and Nakakoji et al. (2002) describes two main characteristics as:

- The community members *identify* themselves with the development of the Open Source software. This identity encourages unity and collaboration on information and expertise. It leads to trust and more importantly, social reputation and peer recognition within their community;
- All members have roles within the community. In order for the community to function productively and to maintain its sustainability over time, diverse member roles are needed. Nakakoji et al. (2002) defines eight roles of community members – passive user, reader, bug reporter, bug fixer, passive de-

veloper, active developer, core member, and project leader. These roles, listed in order from minimal to maximum contribution, have varying influence on the open source software's progress. However, these roles are often not fixed within the communities and can change over time.

For open source communities to be sustainable, diversity in member roles is necessary for the success and evolution of open source software. As advocated by O'Reilly (1999), Cubranic (1999), Schmidt et al. (2001) and Nakakoji et al. (2002), a collaborative and open community can promote remarkable progress in software development. Good examples of kind of OSS success include the Samba project, Linux, Apache web server, and OpenBSD.

I.3. Laws for Software Evolution

Lehman et al. (2001) identified eight rules for software evolution planning and management. These rules were derived from studies of FEAST research projects and OS/360-70 systems between 1969 and 1985. These rules have since been empirically researched by Kemerer and Slaughter (1999), and later revised by Lehman et al. (2001) (Cook, 2000, Lehman, 2001). Table 1 summarises the eight rules.

Lehman classifies software programs into three categories – E-type, P-type, and S-type. However, only E-type is included here as it has the greatest relevance to this research.

- *E-type* programs are systems that are actively in use and are embedded in the real world domain. It also ad-

No	Year	Name	Law
1	1974	Continuing Change	E-type systems must be continually adapted else they become progressively less satisfactory in use.
2	1974	Increasing Complexity	As an E-type system is evolved, its complexity increases unless work is done to maintain or reduce it.
3	1974	Self Regulation	Global E-type system evolution processes are self regulating.
4	1978	Conservation of Organisational Stability	Unless feedback mechanisms are appropriately adjusted, average efficient global activity rate in an evolving E-type system tends to remain constant over product lifetime.
5	1978	Conservation of Familiarity	In general, the incremental growth and long-term growth rate of E-type systems tend to decline.
6	1991	Continuing Growth	The functional capability of E-type systems must be continually increased to maintain user satisfaction over the system life time.
7	1996	Declining Quality	The quality of E-type systems will appear to be declining unless they are rigorously adapted, as required, to take into account changes in the operational environment
8	1996	Feedback System	E-type evolution processes are multi-level, multi-agent feedback systems.

Table 1: Software Evolution Laws. Source (Lehman et al , 2001).

dresses issues such as quality, behaviour in execution, performance, ease of use and changeability.

According to the first law of *Continuing Change*, E-type systems must continually adapt to changes in the environment, including the systems lifecycle, and technical and users requirements. It assists developers to align the systems functions with changing requirements. Without systems-user alignment, the system serves little purpose for users.

According to the second law of *Increasing Complexity*, as the system develops and evolves, the system's complexity increases. This includes increases in the number of elements, increases in the work required to maintain compatibility between interface versions, increases in the potential for error and omissions. Lehman et al. (2001) defines five degrees of systems complexity: application and functional complexity, specification and requirements complexity, architectural complexity, structural complexity. These complexities create difficulties in design, adaptability and systems validation. Therefore, it is necessary for developers to maintain the system and coordinate its activities for upgrades in such a way that these complexities do not reduce system's efficiency or performance.

The third law of *Self-Regulation* states that the systems evolution processes are self-regulating. The rationale for this is that software processes are constantly assessed and evaluated based on their past and present performance. These evaluations are then used as feedback to modify the system where necessary, and an appropriate evolutionary development strategy can be formed. Hence, this cyclical process provides self-regulation to the system evolution process.

The fourth law of *Conservation of Organisational Stability* suggests that appropriate feedback mechanisms are necessary to ensure continuous success in the software development process of the system. This law was first observed in the 1970s. Since then, further observations have been made in the FEAST/1 system and the results neither support nor negate this law. However, Lehman continues to advocate this law, for software evolvability purposes.

According to the fifth law *Conservation of Familiarity*, systems growth will decline over time as the system ages. The basis for this law is that as a system ages, the need for repairs increases, in terms of systems upgrade-ability and compatibility with its current environment. As a result, very little investment is applied to new and innovating projects. This implies the available resources for systems growth is being diverted to maintain and upgrade the existing software, rather than being used for systems enhancements and expansions.

The sixth law of *Continuing Growth*, is clearly distinguishable from the first law of *Continuing Change*, in that the latter reflects the need to *adapt* the system to the real world domain, including its applications and activities used for change. By comparison, the sixth law reflects that all software – its features and functionality can only expand to the boundaries of its current domain. This can create bottlenecks or irritant situations in the current do-

main; hence the system requires continuous growth, including the ability to expand beyond its existing domain, such that it can support new situations and circumstances, if necessary.

The seventh law of *Declining Quality*, follows the fifth and sixth Laws, suggesting that functionalities need to be changed and extended, including the creation of new interfaces and interactions, as the operational domain increases. Operational adaptations are necessary to counteract declines in performance, functionality and reliability, increases in the number of faults, mismatches in operational domains and growing complexities. The lack of resource investments into quality assurance can potentially reduce user satisfaction.

The eighth law of *Feedback System*, underlies the behaviour of all the other seven laws. This law states that software processes are multi-level, multi-loop, multi-agent feedback systems. These systems are constantly modified and extended for improved functionality and/or additional components, thereby meeting the changing needs of the users. This implies the system provide feedback to inform the developers, such that the system continues to evolve. Positive feedback can encourage functional extensions, which can lead to growth and adaptations. Negative feedback has a stabilising influence, whereby internal systems modifications are made to provide internal consistency.

II. PREVIOUS STUDIES ON OPEN SOURCE EVOLUTION

Godfrey et al. (2000) studied Open Source Software Evolution, using Linux as a case example and used the following metrics to measure software evolution for Linux:

- Tracking and recording the number of changes made, as a result of defects in the source files or modules;
- Counting the source lines of codes (SLOC), which will provide a general measurement rate of growth within each module;
- Tracking the number of functions added and plotted over time;
- All measurements taken were plotted against time, rather than version numbers. Plotting against time will determine the growth over time, independent of the release versions;
- In addition to implementation ("*.c") files being measured, header files ("*.h") were also measured.

The results of their study on Linux have indicated strong growth in major sub-systems. The growth patterns in terms of SLOC, source files, and functions are all similar to one another, and all are growing at a "*super linear rate over time*" (Godfrey et al., 2000). The earlier releases grew at a slower rate than later versions. This can be attributed to new functionalities being added rather than code modifications made to existing functionalities. The popularity of Linux in recent years and the consequent user demands have also provided further impetus for Linux to expand its functionalities.

Major growth in sub-systems occurred mainly in drivers for complex hardware devices. When the results from the number of files and SLOC

were cross-referenced, it can be noted that file size increase is attributed to bug fixes and improvements made to existing functionalities. Also, newly created files that were small in size contain new functionalities added to the version. Analysis of ten major sub-systems revealed the greatest and fastest growth occurred in sub-systems that were associated with the network support for Linux.

Mockus et al. (2002), whilst studying the evolution process of the Apache Server and Mozilla software in 1995-1999, provided the following guidelines for the analysis of Open Source Software evolution:

- Sorting and analysing the developers' email list for information containing messages, technical discussions, proposed changes, and automatic notification messages about changes in the code and problem reports;
- Analysing the 'Problem Reporting Database' to measure changes to the code and documentation and individual reports on faults;
- Measuring and examining the source lines of codes (SLOC), to determine changeability for each module, and for the project as a whole.

The results indicate a large community of developers (approximately 400 individuals) who were actively contributing code. Of the 400 people, 182 individuals contribute to 695 problem report changes, while the remaining 249 individuals contributed to 6,092 non-problem reporting changes. Participation of the external community has been greater in bug repairs than in introducing new functionality. All new functions are mainly implemented and maintained by the core team with little external involvement. The productivity of the core team is further observed by increases in SLOC and concurrent version system (CVS) commits with changes to existing code and additions of new code. The study found that Apache continued to exhibit strong community support in terms of active contributions. The results from the Apache case study was compared and contrasted with those from a study of the Mozilla project.

Based on the results of this study, Mockus et al. (2002) derived the following conclusions:

- Open source developments will have a core of developers who control the code bases;
- The core team is not larger than 10 to 15 people and will generate more than 80% of new functionality;
- The community assists in problem reporting, and the core team will respond by repairing faults;
- In open source projects where the number of external contributions never exceed the contributions made by the core team, will result in more maintenance codes rather than new functionality codes;
- Successful open source software developments are generally characterized by developers who are also users of the software;
- OSS developments exhibit very rapid responses to customer problems.

Koch and Schneider studied the growth and evolution of GNOME open source project for building a desktop

environment for users and an application framework based on the GNU network model for software developers. The main metrics they employed were lines of code (LOC) and cumulative LOC, number of persons and time spent on the project, and number of files worked on. They reported a high correlation between total LOC and the number of programmers based on monthly data (Koch and Schneider, 2002).

III. SAMBA CASE STUDY

Samba is a suite of Unix applications, communicating over the SMB (Server Message Block) protocol, with the ability to mediate between Unix and Windows operating systems. The Samba-enabled Unix machines have the capabilities to masquerade as a server on the Windows network, to provide the following services:

- Sharing one or more file systems;
- Sharing printers installed on both the server and its clients;
- Assisting clients with Network Neighbourhood browsing;
- Authenticating clients logging on to the Windows domain;
- Providing and assisting with WINS (Windows Internet Name Service) name server resolution.

Through the provisions of these services, Samba has bridged the gap between the two popular yet very different, operating systems – Windows and Unix. Windows users can now access file and print services irrespective of whether these services are offered by Unix or Windows.

The Samba suite are developed on the basis of two Unix daemons that provide resource sharing between SMB clients (Hertel, 2001, Blair, 1998): smbd daemon provides file and printer sharing services on the SMB network with authentication and authorisation services. It also provides two authentication modes – "share mode" and "user mode", to protect shared and private file and print services with passwords. Samba also provides Domain authentication service, handled by the Domain Controller. The Samba Domain Controller refines the authentication service further, by allowing the user to only log in once, in order to have full access to all of the authorised network services. nmbd daemon provides name resolution and service announcement (browsing) services, in which its role is to advertise the services offered by the Samba server. This daemon has the responsibility of managing and distributing NETBIOS names. NETBIOS is a software program installed in memory that provides an interface between the systems software and the network hardware.

In addition, Samba also provides various utilities. The most popular utilities include smbclient with a shell-based utility similar to the FTP programs in which the client can copy, share, transfer files to and from servers using the SMB network protocol. This includes file and print servers. nmblookup is a NETBIOS name service client for looking up IP addresses and querying remote machines for lists of names, and SWAT is the Samba web administration tool (Hertel, 2001).

III.1. Research Method

III.1.1. Quantitative Measurement

Most of the quantitative data collected on Samba was from Samba's official website (www.samba.org) and StatCVS (statcvs.sourceforge.net) that contain information on current Samba projects under development, previous versions of Samba, archival information on its previous versions, mailing lists, and discussion groups. The data were sorted and analysed. The analysis techniques used were based on previous research techniques employed by researchers in the field of software evolution. (Lehman et al., 2001, Godfrey et al., 2000, Mockus et al., 2002). These researchers have devised both qualitative and quantitative measurements of software evolution, and to varying degrees, concur with one another on their measurement techniques.

The five release versions of the Samba software were evaluated in terms of technical and operational evolvability. These releases were selected because they were stable releases, with their dates of release falling around April every two years. This provided an adequate sample of release versions over time, for comparison and assessments be made on Samba's evolution, in the 1995 to 2002 period. The technique used for data analysis include:

- Counting the SLOC (source lines of codes) in each module of the five versions, using Perl scripting. The SLOC measurements were graphically plotted against time, to evaluate whether SLOC have increased with each new release version;
- Analysing functionality by counting the number of additional functions within each module and version, then comparing the results. The functionality measurements taken are graphically represented against time;
- Analysing the number of bug fixes and patches in each new released version. This measurement was plotted against the version number to determine whether bug fixes have increased or decreased with each new release version. Growth in the number of bug reports and repairs sent by the Samba community could very well imply that the code is bug-ridden and of a poor quality. However, we operate under the assumption that any large, complex software code will have bugs and one of the key strengths of the open source model is its ability to discover a good proportion of them through the efforts of the community. It is in this sense that we use the increase in bug fixes and reports as an indication of the growing effectiveness and evolution of the community;
- Evaluating the size of the Samba team and their source code contribution. This determined the contributions made by each team member and assist in the examination of the Samba team structure, in relation to their individual contribution to Samba's progress;
- Analysing mailing lists and discussion groups at the time of the releases of the five different versions of Samba. The size of the mailing lists and discussion groups indica-

te Samba's popularity and growth, and determined whether there exists a complementary relationship between Samba's growth and its community involvement.

This analysis helped us track and scrutinise Samba's evolution. In order to further assess Samba's future software evolvability, Lehman et al.'s (2001) rules for software evolvability were tested against the Samba experience. The main drivers of Samba evolution were identified and documented.

III.1.2. Qualitative Measurement

In terms of community solidarity and Samba coordination of its projects, we interviewed the members of the core Samba team to analyse and evaluate the methodology the Samba team used to coordinate and integrate their projects. Interviews were conducted with three of the key developers who were involved at every stage of the Samba project. The interview data provided deep insights into the structure of the Samba community and to assess Samba project success in relation to community collaboration.

The combination of qualitative and quantitative techniques were employed to ensure that that the assessment of Samba's growth and evolution is thorough and comprehensive and constitutes an improvement over previous studies. The interview data enabled us to fill the gaps in our understanding of Samba evolution, obtain clarifications, and to triangulate on the phenomenon under scrutiny.

III.2. Results: Quantitative Analysis

Table 2 shows that SLOC measure showed strong growth in Samba's release versions especially during the periods that coincided with Microsoft Windows releases. To specifically mark where the growth has occurred, each modular component of Samba was analysed separately to determine the source of the growth. It was discovered the lib (architecture-independent library code), smbd (server code for authentication and authorization),

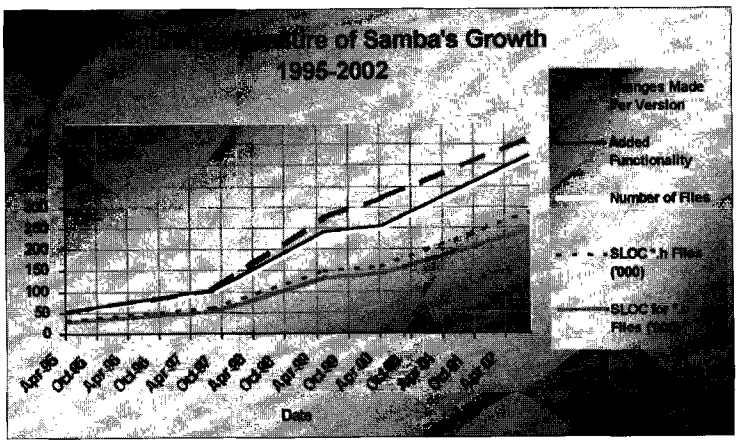

Table 2: Quantitative Measure of Samba's Growth.

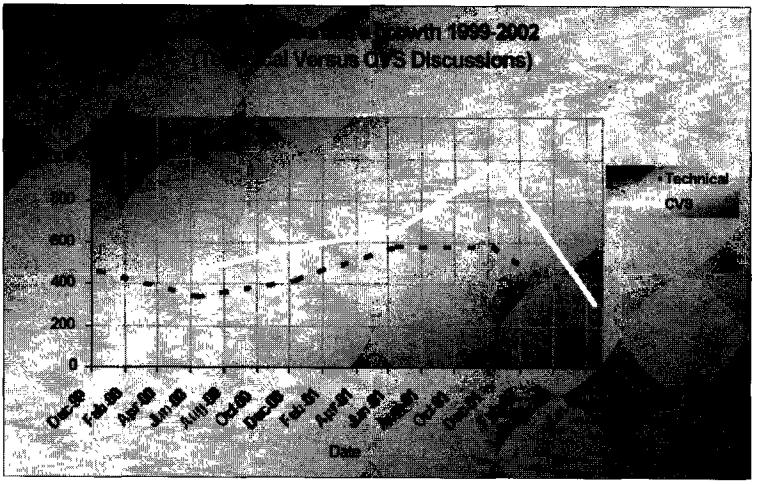

Table 3: Measuring Samba's Discussion Activity (Technical versus CVS).

and param (parameter definition code) modules had the greatest and most consistent growth than all other modules. In contrast, the printing, utils and client modules exhibited erratic growth rates. This can be attributed to SLOC being transferred from one module to the next. Since its creation, Samba has moved to become more modular. In order to achieve this, it was necessary to move code between modules, where appropriate.

SLOC has been criticised as a flawed measure in that SLOC can be transferred between modules and good developers aim for the minimal amount of code to produce the best quality code. In order to supplement the SLOC measurements the number of files measurement is applied to account for SLOC movement between modules. The results indicate there has been a progressive growth in the number of files in each release version. Even though the number of files does not, in itself, provide a complete measure of Samba's growth, it does provide indirect supportive evidence.

In order to better substantiate Samba's growth, functionality measurements were adopted for comparisons between versions and modules. Such comparisons investigated not only whether Samba has evolved since its original release, but to identify the source of the growth. The results illustrate that the functionalities added per release version were relatively equal between versions. However, the number of changes made to the existing code in the form of bugs and security patches were quite high and fluctuating. This suggests that improvements and repairs made to the existing code had greater priority and were more time consuming than adding extra functionality. This can be viewed as a strength because new functionality should not be added to a faulty and unstable working version.

Functionality provides the clearest measure of Samba's evolution and growth. However, additional supplementary results from SLOC and the number of files provide complementary evidence of Samba's evolution.

Overall, the results indicate that Samba has grown and evolved in terms of SLOC, number of files, and in functionality. However, even though the data analysis indicates a steady and gradual growth over the period 1995 to 2002, there were some periods of the Samba project that have experienced erratic growth. This erratic behaviour suggests that Samba's progress and evolution to its current stage was not without obstacles (discussed in greater detail in section 3.5). In order to evaluate the evolution process of Samba and the obstacles encountered, Samba's community was studied.

Samba's community was examined to determine the amount of active and passive contribution it has made to Samba's evolution. The Samba community contribution was quantitatively measured in terms of CVS and technical discussions (see Table 3) and qualitatively validated through interview sessions with the Samba team members. CVS discussions are obtained from Samba's global message boards in which any member of the community has the ability to post or respond to any of the (typically code-related) messages. In contrast, technical discussion messages are obtained from Samba's private message board on which Samba team members discuss core technical issues. The quantitative results highlight both CVS and technical discussions have been progressively and gradually increasing since Samba's debut. There is also strong evidence of a pattern of active contribution from the greater community.

However, the results also indicate that during the period following December 2001, both CVS and technical discussions had begun to decline pointing to an increasing passive contribution to the Samba project. This suggests the composition of the community has evolved from that of primarily comprising of active developers to one filled with passive and general users. Despite the community's growth, the ratio of active developers to passive users has declined in the number of active contributors.

In conclusion, the Samba project has exhibited growth in every area that has been measured – both qualitatively and quantitatively – including SLOC, number of files, functionality, and the Samba community via analysis of the discussions and interview sessions. Both qualitative and quantitative techniques were adopted with the view that these measurements will overlap and validate each other.

III.3. Qualitative Analysis

Samba's development process is loosely formed and structured. All projects within the Samba team are internally discussed online using IRC chat, email messages, and the Samba team's technical discussion board. A general consensus is formed by the Samba team regarding the technical requirements and specifications of the project. All internal Samba projects are subjected to a "loose" deadline for staged incremental releases. Projects external to the Samba team do not require approval, however, discussions with the Samba team would be appreciated and future assistance from the Samba team will be given with a greater level of ease and knowledge.

Projects internal to the Samba team are undertaken and prioritised based on:

- The developers' interest in addressing particular issues;
- The needs of the mailing lists, including compatibility and functionality issues;
- The needs of the corporate sponsors, including compatibility and functionality issues.

Technical conflicts that arise are usually resolved with the policy of "code speaks louder than words". The developer who can first implement the functionality will immediately prove that such functionality is possible to implement. Problems are usually discussed online through electronic means such as email and discussion boards.

In the early days of Samba, all code contributions were sent to Andrew Tridgell or Jeremy Allison, the only two members of the Samba team with complete access to all of the Samba source codes. The coordination and integration of Samba projects were entirely organised by them. In recent years, the role of integration has been relinquished to the whole Samba team. As a result, the general community may contribute code by electronically sending their code to any member of the Samba team, and the team will determine whether the codes should be integrated with the new version of Samba.

Every Samba release version has a release coordinator. The role of the release coordinator is to ensure the complete integration of all code contributions and its maintainability as a whole, and that the Samba version is stable before release. This implies that before the version is released to the general community, Samba will undergo quality assurance and testing to check that the version is stable, reliable, and maintainable.

All code contributions sent to Samba are assessed on the following basis:

- Whether the code contributed is good code, in terms of structure and interoperability;
- Whether the code can be easily integrated and is potentially maintainable. Hence, it helps if the source code is modular, and it can be attached to the existing version as a module;
- Whether code additions and modifications are possible once the contributed code has been integrated with the existing module.

However, code contributions made by the internal Samba members are deemed reliable, and hence, their code contributions are more trusted and are more readily integrated into the existing Samba version than general contributions.

Once the version is released to the open source community, the Samba team progresses to the next project. The product release also provides feedback opportunities from the general public about the product's functionality and operability. The open source community submits reports on the product's defects and bugs that will enable the Samba team to make modifications to the current release where necessary. The feedback opportunities attract additional testers of the software which is one of the unique and beneficial aspects of open source software.

Figure 1 presents a model of Samba's development process. It appears to en-

capsulate most of the activities undertaken to produce a release version. The model is also mapped to the law of software evolvability presented by Lehman et al. (2001) and discussed in the section 3.4 below.

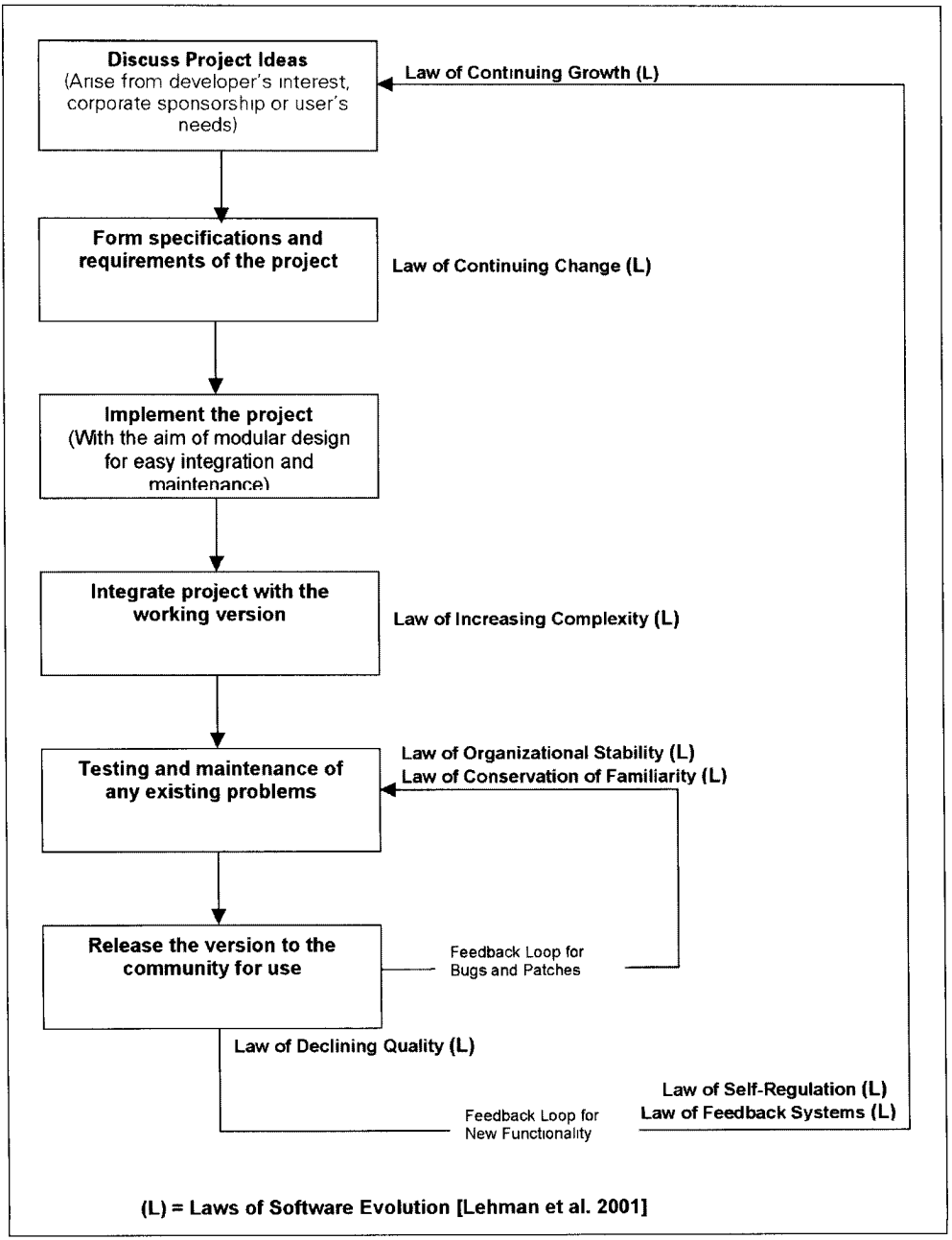

Figure 1: Framework for Samba's Development Process.

III.4. Discussion

In this section we evaluate Samba evolution against the Laws of Software Evolution as outlined by Lehman et al. (2001).

In terms of the *Law of Continuing Change,* Samba's added functionality (shown in Table 1), Samba's discussion – technical versus CVS (shown in Table 2), and Samba version changes (shown in Table 2) and the foregoing discussion in section 3.2 indicate that adaptations were made to each version of Samba suggesting that Samba has adapted to the changing needs of its user community.

Regarding the *Law of Increasing Complexity,* both qualitative and quantitative measurements have revealed Samba has indeed evolved. From the data obtained from the interview sessions reported above, there is evidence of Samba moving towards greater modularity to combat maintenance complexities as Samba grows.

Concerning the *Law of Self-Regulation,* Samba is regulated by its community. Samba's efficiency and operability depends on the software evaluations (including performance and faults issues) from the community. These feedback mechanisms are used as input for future modifications and new projects.

In terms of the *Law of Conservation of Organisational Stability* and the *Law of Feedback System,* Samba's feedback mechanisms involve the Samba community providing critical evaluations on its current release version, including submitting patches and suggesting possible functional extensions. The Samba team uses the bug tracking system for reporting bug problems. This global feedback allows the current version to be improved and provides greater stability of the current version.

The *Law of Conservation of Familiarity* assumes that as Samba ages, there will be a greater need to maintain the existing code than for introducing additional functionality. Evidence of the validity of this law with respect to the Samba project is mixed at best. It can be seen from Table 2 that the number of changes made exceed the amount of functionality added. However, to ensure Samba continues to evolve, new functionality needed to be continuously introduced. During our interview, Andrew Tridgell highlighted the importance of undertaking several potential future projects to enhance the current version. Samba currently deals with only a small section of a very large SMB protocol; hence Samba has the potential to encapsulate more of the SMB protocol. As for Lehman et al.'s *Law of Continuing Growth,* which states that software must enhance its functional capability to maintain high levels of user satisfaction, we have produced considerable evidence of Samba continuously adding functionality to respond to the users' changing needs

The *Law of Declining Quality* highlights that Samba must be continually maintained and its functionality upgraded in order to counteract the declining quality in performance, functionality and reliability as Samba ages. This has certainly been the case with strong evidence of Samba responding to expeditiously to new releases of Windows versions. Overall, Samba

seems to satisfy all but one of the laws of software evolution proposed by Lehman et al. (2001).

III.5. Drivers of the Samba project

The qualitative and quantitative results and analysis of Samba have shown that Samba has progressively improved in terms of SLOC and functionality. With the assistance and support from an active community, Samba has achieved growth. During the period of 1995 to 2002, Samba had been continuously and consistently evolving. In this section we draw on the qualitatitive data gathered to analyse the key drivers of Samba's evolution.

Our investigation of Samba has helped us to identify and explore some of the key drivers of Samba's evolution. The more important ones are:

- Community dedication in providing support, interest and sponsorship of Samba's development;
- Community or commercial requests for particular functions;
- Core team dedication and personal motivation;
- The new releases of Microsoft Windows that creates interoperability issues;
- The low cost effectiveness of Samba's operations;
- Samba's feedback system and frequent beta releases;
- Greater need and demand for the Samba software.

The Samba community is one of the drivers of Samba's evolution. Community support and sponsorship has provided financial support in its contribution to Samba's evolution. These sponsorships are usually conditional whereby the corporate sponsor requests that particular Samba-related projects be undertaken, according to the needs of the corporate sponsor. Hence, sponsorship encourages Samba to remain updated and to meet the demands of its community.

Samba's community as of 2002 comprises of 10% developers and 90% general users. The number of developers has gradually increased over the last ten years; however, the user community has increased at a faster rate than the developer community. Community requests to the Samba team indicates an increasing popularity and interest in Samba and this fuels the core team to adapt Samba to the changing needs of its users. This includes maintenance and upgrades of current functions, and the addition of new functionalities, as dictated by the community's demands and interests. Keen interest in Samba have also motivated developers to contribute code, especially those seeking peer recognition and "Open Source fame", hence, adding to the popularity and friendly "code competitions" within the community.

The core team's continuous dedication and cohesion in Samba's development has driven Samba to be greater coordinated in its tasks. These tasks include maintenance and testing of existing code, improving existing functionalities, coordinating new Samba-related projects, and integration of new features with the working version of Samba. Samba has also moved to become more modular, as Samba's popularity grew and the OS community were more acti-

vely involved with code contributions. Hence, Samba's modularity has simplified the maintenance process and promoted projects that can be simply attached to the existing Samba version, independent of any modifications made to other sections of the working version.

Microsoft releases of Windows 98, 2000 and NT remains one of the greatest driving forces of Samba's evolution. From our data analysis, it was observed that Samba exhibited the greatest growth in SLOC, and number of files in the versions that coincide with Microsoft Windows releases. As reported on April 3rd 2000 by InformationWeek, the Microsoft release of Windows 2000 has created interoperability problems for Samba, whereby Microsoft has made modifications to its Remote Procedure Call (RPC) authentication protocol, and left the Samba client unable to fully interoperate with Windows 2000. This example of interoperability problems between Samba and Windows illustrates the need for the Samba team to make necessary adjustments to its source codes to fully interoperate with Windows. It is also important that any modifications made have both backward- and forward-compatibility with previous version of Windows and Samba. Thus, Microsoft Windows releases remains to be one of the biggest drivers of Samba's evolution.

Cost plays a role in driving Samba's evolution. Since Samba relies on a pool of dedicated volunteer developers, it is essential that Samba's progress and evolution be structured where code maintenance is kept to a minimum. This has driven the Samba team to construct its design to be modular so that any changes within modules occur locally within that module with little effect on other modules. This allows Samba to easily expand its functionality within its current structure and also to expand beyond its current domain over time. Another factor in keeping low costs is by encouraging all communications and contributions relating to Samba to be done over the internet. This provides greater community collaboration and provides easy distribution of Samba code to interested parties.

The broader feedback system enables the open source community to comprehensively scrutinize Samba's performance, reliability, and functionality. The existence of such a broad and diverse feedback system has pushed Samba to evolve through a systematic process of bug patches and fault repairs.

We conclude that the variety drivers of Samba's evolution – ranging from community requests to feedback systems – have encouraged growth in Samba and strengthened the evolutionary processes that are followed by the Samba team. Figure 2 illustrates the drivers of Samba's evolution. These drivers promote evolutionary processes that are used to improve the current working version of Samba. The results of applying these evolutionary processes have contributed to increases in SLOC, community discussions, functionalities and number of files thereby facilitating successful evolution of Samba software.

IV. CONCLUSION

We studied the dynamics and evolution of open source software using Samba as a case study employing re-

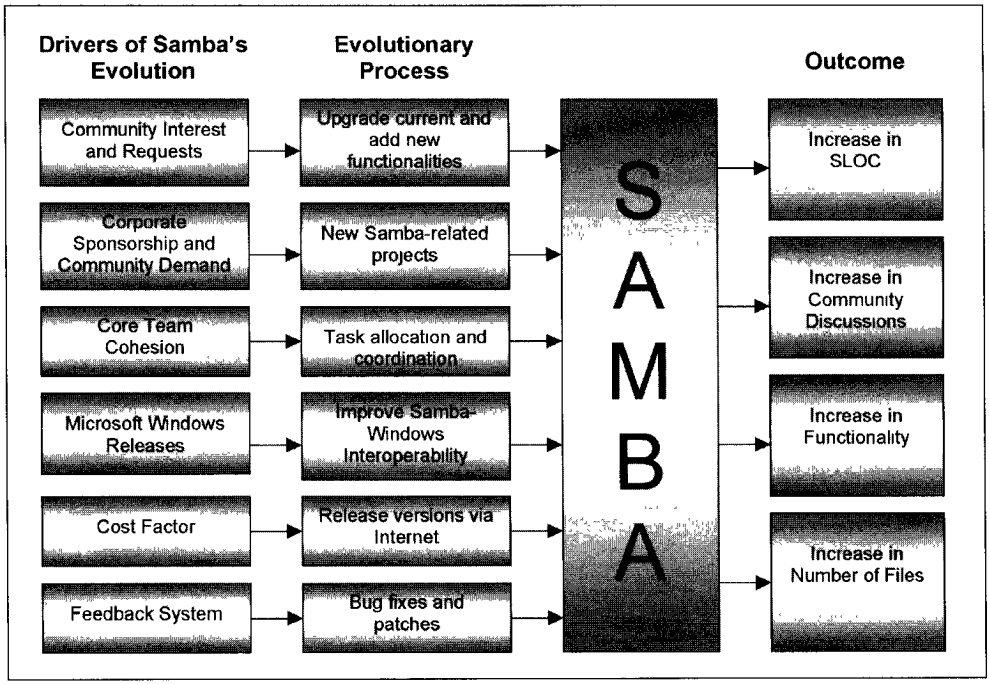

Figure 2: Samba's evolution.

search methods previously used by other researchers in the field. Both qualitative and quantitative data were analysed to develop a comprehensive understanding of the patterns of evolution of Samba. These patterns were evaluated against the laws of software evolvability proposed by Lehman et al. (2001) and found to be generally consistent with them. The key influences on and drivers of Samba software growth and evolution and OSS were identified and analysed.

The combination of quantitative data analysis with insights gained from the interview data gathered from the core Samba team enables us to document and model the Samba evolution with greater specificity than was possible in previous studies. The latter also helped us to explore the wide range of internal and external influences on a typical OSS project and how they drive the evolution of the software.

V. REFERENCES

Blair, J (1998), "Introducing Samba", *the Linux Journal Review*, July 1998, http://www linuxgazette.com/issue36/blair.html

Browne, C. (1998), Linux and Decentralised Development, First Monday, 3(3), Available at http://firstmonday.dk/issues/issue3_3/browne/index.html.

Brown, J.S. & Duguid, P. (2000), "The Social Life of Information", *Harvard Business School Press*, Boston USA.

Cook, S., Ji, H. & Harrison, R. (2000), "Software Evolution and Software Evolvability", *http://www.personal.rdg.ac.uk/sis99scc/papers/Harrison_00a.pdf*

Cubranic, D. (1999), "Coordinating Open Source Software Development", *Procee-*

dings on the 7th Workshop on Coordinating Distributed Software Projects, IEEE Digital Library, 1999.

Feller, J. and Fitzgerald, B. (2000), "A Framework Analysis of the Open Source Software Development Paradigm", *Proceedings of the 21st Annual International Conference on Information Systems*, Brisbane, Australia, December 2000.

Gacek, C. Lawrie, T. & Arief, B. (2001), "The Many Meanings of Open Source", Citeseer Digital Library, http://citeseer.nj.nec.com/485228.html.

Godfrey, M., and Tu, Q. (2000), "Evolution in Open source Software: A Case Study", *Proceedings of the International Conference on Software Maintenance*, San Hose California.

Hertel, C. (2001), "Samba: An Introduction", http://www.samba.org

Koch, S. and Schneider, G. (2002), "Effort, Cooperation, and Coordination in an Opne Source Software Project: GNOME", *Information Systems Journal*, 12, p. 27-42.

Lehman, M.M. & Ramil, J. (2001), "Rules and Tools for Software Evolution Planning and Management", special issue on Software Management, volume 11, *Annals of Software Engineering*, Autumn 2001.

Markus, M.L., Manville, B., and Agres, E.C. (2000), What makes the virtual organisation work? *Sloan Management Review*, Vol. 42, n°1, p. 13-26.

Mockus, A., Fielding, R.T. & Herbsleb, J. (2002), "Two Case Studies of Open Source Software Development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, Vol. 11, n°3, p. 109-346.

Moody, G. (2001), *Rebel code – Inside Linux and the Open Source Movement*, Cambridge, MA: Perseus Publishing.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. & Ye, Y. (2002), "Evolution Patterns of Open Source Software Systems and Communities", *Proceedings of the workshop on principles of software evolution*, Orlando Florida, 2002.

O'Reilly, T. (1999), "Lessons from Open Source Software Development", *Communications of ACM*, Vol. 42, n°4, April 1999.

Raymond, E.S. (2001), *The Cathedral and the Bazaar: Musings on Open Source and Linux by an Accidental Revolutionary*, Sebastopol, CA: O'Reilly.

Scacchi, W. (2001), "Software Development Practices in Open Software Development Communities", *1st Workshop on Open Source Software Engineering*, Toronto, Ontario, May 2001.

Scacchi, W. (2002), "Understanding Requirements for Developing Open Source Software Systems", *IEEE Proceedings on Software*, 2002.

Schmidt, D.C. & Porter, A. (2001), "Leveraging Open-Source Processes to Improve the Quality and Performance of Open-Source Software", *1st Workshop on Open Source Software Engineering*, ICSE 23, Toronto, Canada, May 15, 2001

Swoyer, S. (2000), "Windows 2000 complicates interoperability for Samba", Issue 780, April 3rd, 2000, *InformationWeek*, Proquest Computing.

StatCVS (2002), Sourceforge Project at http://statcvs.sourceforge.net/

Wang, H. & Wang, C. (2001), "Open Source Software Adoption: A Status Report", *IEEE Software*, Vol. 18, n°2, p. 90-95, March 2001.

Wenger, E. (1998), "Communities of Practice Learning as a Social System", Published in the "*Systems Thinker*", June 1998, http://www.co-i-l.com/coil/knowledge-garden/cop/lss.shtml

Yamauchi, Y., Yokozawa, M., Shinohara, T. & Ishida, T. (2000), "Collaboration with Lean Media: How Open Source Software Succeeds", *Proceeding of the ACM Conference on Computer Supported Cooperative Work*, Philadelphia USA, p. 329-338.