Quantitative Analysis of

Open Source Software Projects

Ismail Ari
University of California Santa Cruz
ari@cse.ucsc.edu

Abstract

The Open Source Software (OSS) model and the GNU Public License (GPL) have recently been challenged by Microsoft's Commercial Software Model (CSM) and "shared software" plan. With the discussions turning into debates and rivalry [1], it has become much more critical to understand the nature of different software ecosystems.

This paper reviews different software development models and aims to shed more light on the mechanisms of OSS. As examples, we investigate two communities, one developing a technical requirements specification and driven by commercial motivations and the other developing a network simulator and driven by academic motivations. For these two communities, our findings indicate strong similarities between the OSS model and large program development models. With the lessons learned and metrics obtained from the analysis of these two communities we develop a simulation environment that would enable social scenarios to be generated and projections to be made about the productivity of ongoing OSS projects.

1 Introduction

Although convinced with the strengths of Open Source Software (OSS), many software companies are still reluctant to fully deploy their resources into OSS development, because they do not want to jeopardize their business by shifting into an ad-hoc and mystical domain. In fact, OSS advocates have contributed a great deal to the mysticisms by making religious comments about OSS concept. Developing quantitative metrics that will render OSS a measurable concept will melt the ice between businesses and OSS. The recent challenge from Microsoft to OSS communities has proven the importance of quantification of the OSS phenomenon. Any academic and scientific investigation that will contribute to clarification of the OSS concept will also help define the future of the software ecosystem.

Five key elements constitute Microsoft's Commercial Software Model (CSM) [2]. Strong community of developers, standards promoting collaboration and interoperability, a profitable business model, research and development (R&D) investments and finally a licensing model that does not jeopardize the intellectual property (IP) rights. There is no doubt about the strength of a networked and collaborating community of developers, as this fact has proven itself over many years and many projects such as Linux, BSD, Perl and Apache. The concerns and disagreements are centered on the "openness" of the source code and the licensing that enforces the degree of this openness.

GPL intends to guarantee the freedom to share and change free software and therefore makes restrictions that forbid anyone to deny these restrictions. According to the terms and conditions item #0 of GPL [3], GPL license applies to any program or derivative program (or work) based on the original program distributed under the terms of GPL. This notion is also called "copyleft" [3]- a term coined by Richard Stallman, the founder of Free Software Foundation (FSF). The requirements on derivative work have triggered the Microsoft executives to describe GPL as "viral" and "infecting" the derivative programs, thus threatening the businesses wishing to obtain value from their value-added intellectual property. At the same time the Berkeley Software Distributing (BSD) license allowing programmers to use, modify and redistribute the codes, without requiring them to publish the new source code or allow royalty-free redistribution has been favored by Microsoft. This approach sounds similar to the "logs vs. lumber" analogy [4] describing the difference between OSS and commercial products. Logs represent the low level or system level code and lumber represents the value added to base code in the form of pre-built binaries, documentation and interfaces.

"Shared source" philosophy [5] is more constrained than the two licenses mentioned above. It only allows companies to share source code with their business partners and customers. The goal of this philosophy is to enable collaboration and research *without destroying the software businesses built around a strong intellectual property (IP) base*. A detailed comparison of currently existing licensing practices can be found in [6].

The business end of different software ecosystems is out of the scope of our research. However, these discussions make it clear that unless the quantification and clarification of the OSS domain is made through academic and scientific research, fanaticism will dominate the future of software process modeling analysis. This paper aims to contribute to the quantification of OSS, by rigorous analysis of two large-scale software development projects with a large customer base and market potential. We analyze metrics defined by some of the previous research [7] and propose new metrics.

In section2, we will review different approaches in the previous OSS research. In section 3, we will describe our preferred methodology for analyzing "open" project developments. In section 4, we will briefly introduce the two "open" projects we will be looking at. In Section 5, we give present our finding from the mail and change history archive analysis. In section 6, we propose a preliminary simulation model for OSS and we conclude in Section 7.

2 Background

The previous research on OSS has generally focused on the psychological aspects of collaborative development. Understanding the "hacker culture" [4] has been set as the goal. Some claimed that "hacker culture" was driven by reputation-based and individualistic motivations, while others responded by claiming that OSS was purely altruistic and egoless.

Whether the spirit of the hacker culture is the first or the second or a mixture of both some questions will still need to be answered: "What projects are or are *not* suited to OSS development? How do we create and sustain development communities? How do we resolve clashes? Will there be support for the product of OSS and will it last?" [8]. Many of these and other similar social concerns are answered in detail in Raymond's famous Cathedral and the Bazaar [9].

In the previous research, the methodology and product quality of OSS have also been described qualitatively by similar phrases over and over again. To quote a few, the proponents say "OSS has capacity to compete successfully and even displace commercial development methods; Rules of OSS are different and these rules threaten to do it faster, better and cheaper; OSS has proven to produce software of high quality, functionality and wide deployment; People chose and undertake the work and work is not assigned [7]; Given enough eyeballs, all bugs are shallow [9]". The opponents or people with concerns say "There is no explicit system level design or no design; No project plan, schedule or list of deliverables" [7].

What we really need is to understand and quantify the mechanisms and driving forces of OSS, so that we can:

- use the same mechanisms to structure the commercial projects,
- quickly obtain or create the missing ingredients to speed up new OSS projects,
- understand what role we can take and how much commitment we will have to make in an OSS project,
- how fast we can get an initial working prototype from an OSS given a fixed budget.

3 Methodology

One pitfall to avoid when comparing commercial and OSS projects is the assumption that the differences are only due to the nature of the development styles (i.e. assuming similar products are the same and whatever difference there is, it results from the development style). Unless strict control groups are enforced (such as same or similarly skilled group of people starting with the same requirements, but playing with different rules) the variations will be affected by various other parameters.

We will chose to two "open" development projects with similar demography, but driven by different forces. We do not aim to compare and contrast these two projects, but use their motivational and structural variations as a means to factor out the "invariants" of OSS. Using the lessons learned we hope to come up with a simulation model that could facilitate powerful analysis of OSS projects by enabling scenarios to be tested and projections to be made.

The careful reader has recognized that we do not say, *"open source"*, because in our first example the project artifact is a "requirements specification" (a very technical document encapsulating 2-3 different protocols to support a new architecture called Internet SCSI or storage over IP) rather than source code. We believe, it is valid to assume this project as "open" (source/artifact/results), since first of all the implementation was going hand in hand with the specification development and second any bug fix in the code due to ambiguity in the specification was being an immediate fix, either modification or addition, in the specification, too. Our assumption should be reasonable for the statistical software engineering purposes in this document that aim to find general trends within OSS as large collaborating communities.

3.1 E-mail list archives

The information about open projects can be obtained from two main sources. First is the product of the open source, either the source code or the document produced. However, this is a static source that only gives the snapshot of the latest state. Product investigation only enables analysis on the quality of the product. To gather more *dynamic information* we have to go through the stages of development and the communication between the peers. This type of dynamic information can be found in e-mail archives, which have also been the base of analysis for previous quantitative OSS analysis.

There are tools specialized for e-mail archiving and indexing. MHonArc [10] is one such tool that converts e-mail archives into indexed HTML files. MHonArc is written in Perl and many e-mail archives published online use this tool. The tool is executed over mailboxes saved as huge text files and converts them into multiple html pages as many as the number of e-mails in the mailbox and indexed optionally by thread, author or subject.

3.2 Can we fully automate mailbox analysis process?

It would be useful if we could find repeatable methods that would enable analysis of many mail archives from different technical communities. However, if generic methods are to be obtained one has to limit the procedure to the generic things found in all mail archives. We have noticed that this boils down to the mail headers with fields like: *Subject, from (author), to, reply-to, timestamps etc.* Body text would be totally different in different archives. The question is whether header information is enough or not. But, before that we encountered other problems.

Perl scripts need regular expressions or rules or pattern to operate over the bulk of the data. However, to depend on automated pattern matching process, the researcher at least needs to have *some* idea about the system. We initially tried using Perl scripts similar to the one in Fig.1 to automate our statistical analysis. After this Perl parsing we have seen that there were unimaginably many authors or **From** field possibilities such as:

< Firstname Lastname >, < firstname, middle initial, lastname >, <F.L.>, < Mail System Internal >, etc.

As we kept adding different possibilities into our parsing scripts the problem became more and more challenging because the variations were almost endless. It turned out the same people were using multiple e-mail addresses, upto four in some cases, to sent mail to the same list, which was also causing great ambiguity for the scripts. The effort to capture all possibilities were never enough.

```
#!/usr/bin/perl
$fromcount = 0;

while(<>) {
 if(m/^From:\s*(\w+)\s*(\w+)\s*<(\S*)>/) {
    $fromlist{$2} = $1;
    $fromcount{$2}[0]++;
 }
}
$total =0;
foreach $key (sort keys %fromlist){
  print "$key, $fromlist{$key}, $fromcount{$key}[0]\n";
  $total += $fromcount{$key}[0];
}
print "Total: $total\n";
```

Figure 1: First naïve Perl script that attempts to parse the e-mail archives based on *from* field.

Similar difficulties were encountered for the analysis of the subjects, titles etc. Recognizing that the scripts could lead us to wrong results, we started looking into the mails conventionally to get some patterns to be used in the scripting for future research. As would be the case for any other social system, for a fully automated analysis of OSS communities through e-mail archives, artificial intelligence (AI) has to be employed. Otherwise, static scripts will always remain one step away from understanding the endless idiosyncrasies that could be generated by humans.

3.3 Use of IS Architecture (ISA) as an OSS analysis technique

Zachman's IS architecture (ISA) [11],[12] is a framework for analyzing the OSS development approach. According to this approach the questions *what* (type of project), *how* (is project organized, tools used) and *when/where* (temporal and spatial/geographical dimensions) should be asked to categorize different IS architectures and *who* (agents of change, individuals, companies, users), *why* (assumptions, technological, economical, social motivations) for the logical completeness and detailing. In [11], Feller asks these questions for famous OSS (Linux, Apache etc.) projects. We describe the demography of our "open" project selections in section 4 and make their quantification in section 5, thus inherently answering the questions above.

4 Sample Open Projects

Today open source usually means one of the following major open source projects: Linux, FreeBSD, Unix, Apache Web Server, Perl, Tcl, Python, Sendmail mail server, Mozilla [4]. We will try to look at two other successful projects.

4.1 Internet SCSCI (ISCSI) Project[1]

ISCSI project ([13],[14]) mailing list has approximately 250 active members[2]. The actual number of people working on the project is conjecture to be at least 2-3 times more than this number, since some people act as representatives for their groups in the mailing list.

ISCSI project started in January 2000 as Internet Engineering Task Force (IETF) Birds of Feather (BoF) group. The first ISCSI requirements draft was released in Feb 2000. As the project evolved and more people started joining, auxiliary technical specifications were also produced. The BoF group soon became an official IETF Working Group and internally defined 2-3 more subgroups to focus on different branches of the specification (security, management, fibre-channel interoperability). As of 02/20/2000, the ISCSI project email list archive composed of 3434 messages that formed a tarball of 32MBytes of pure text.

The *demography* of the group can be described as: researchers from various companies and universities. What makes this group interesting is that most attendants are academically oriented people with similar backgrounds, but this time driven with commercial motivations and forces. The demography of this group is very similar to the demography of the network simulator project we will be looking at, where the motivations will be purely research oriented.

4.2 Network Simulator

"Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks [15]." The history of ns is closely related to the history of networking protocols especially TCP/IP. BSD TCP/IP versions and release years are given in Table 1. Most TCP versions appeared in the first few releases of NS. Later versions of TCP and numerous other modules related to Local Area Networks (LAN), Wide Area Networks (WAN), flow control, congestion control, web, multicast, real-time, wireless, ad-hoc and satellite networking were added to ns as their real life counterparts were being developed. Some protocols were even implemented and tested in ns before they were embedded in real systems.

[1] Also called IP Storage (IPS) or Storage over IP (SIP).
[2] As of April 2001. People who have not send any e-mails to the list were not counted, since do not appear in the archive.

Version	Importance	BSD	NS
4.2BSD	first TCP/IP	1983	1988 (NEST)
4.3BSD	TCP improvements	1986	1989 (REAL)
4.3BSD-Tahoe	ss, cwnd, fast rxmit	1988	1995 (VINT-ns1)
4.3BSD-Reno	fast recov., fast path, hdr compr.	1990	1995 (VINT-ns1)
4.4BSD	mcast, LFN	1993	1995 (VINT-ns1)
4.4BSD-Lite	Reno fast recov., fast path, hdr compr.	1994	1997 (NS-ns2)
TCP Vegas	Throughput enhancements	1994	1997
TCP-SACK	Throughput enhancements	1994	To-date

Table1: Evolution of the ns network simulator parallel to the evolution of TCP [16].

The size of Network Simulator (NS) is probably bigger than many other famous OSS projects. In version 2.1b8a the total lines of code (LOC) has reached up to approximately 250,000 lines written in 3 different programming languages (C++, C, and TCL). The uncompiled size of the ns-allinone package that includes graphics and network animation tools in addition to the network simulation code is around 193MBs and easily reaches 300MBs when compiled.

These two communities introduced in this section, have their own version of driving forces that speed up the process of addition of modules and changes being made to the open source code or open requirements specifications. For research oriented source code the critical due dates are usually the *due dates of conferences*. Commercially oriented open developments tend to be driven by the approaching *business fairs* and *business showcases* where many companies are forced to do novel demonstrations to preserve their market shares.

5 Results

In this section, we analyze metrics defined in previous research and propose new metrics. Apache project analysis [7] has focused on the following metrics: core team size, code ownership, productivity, defect density, problem resolution interval (level of support users receive), PR (problem report) changes and non-PR changes. To summarize a few of their results, they found that the core group consisted of approximately 15 people, 182 people contributed to 695 PR changes and 249 people to 6092 non-PR changes.

5.1 Metrics that relate to the roles taken in OSS communities

We start our analysis with the cumulative distribution of contribution based on number of e-mails (Fig.2). Although we do not conclude that a person's activity on the mailing list is the direct measure of effective contribution, due to the immense about of technical information exchange through these e-mails, there is most likely a linear relation between them.

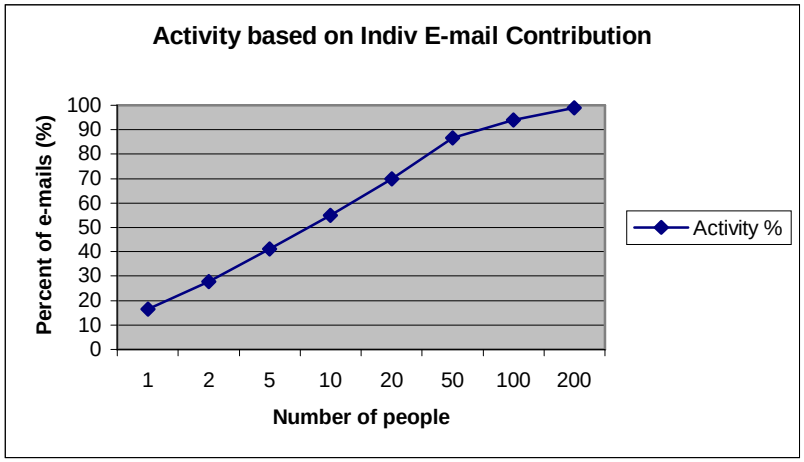


Figure2: The cumulative distribution of contributions to the e-mail archive via comments, bug fixes etc. Note that the number of people is logarithmic showing that majority of the changes come from a small percentage.

Fig.2 shows that almost 80% of the e-mails were submitted by 20-30 people[3]. This is approximately 10% of the total number of people on the list. An investigation showed that these were the people whose names also appear on most of the specifications produced by the group. Therefore, it is possible to define this 10% as the core team of developers. Result of this analysis is consistent with the results found for Apache project analysis [7].

We also have to look at the distribution based on the institutions. This is important because we will see that the *core size* decreases drastically when different people from the same company or institution are considered and counted as one.

Fig.3 shows the 18 institutions that contribute 80% (~2847/3500) of the total email activity in the ISCSI list. Among these 18, e-mails sent by first 3 companies add up to 50% (40% of total) and first 6 companies to 75%. (60% of the total). This analysis proves that looking at individuals may not be representative enough for getting the big picture on the dominant players. Core size as defined by the companies, also has implications about the market share these companies are willing to take with the introduction of this new technology.

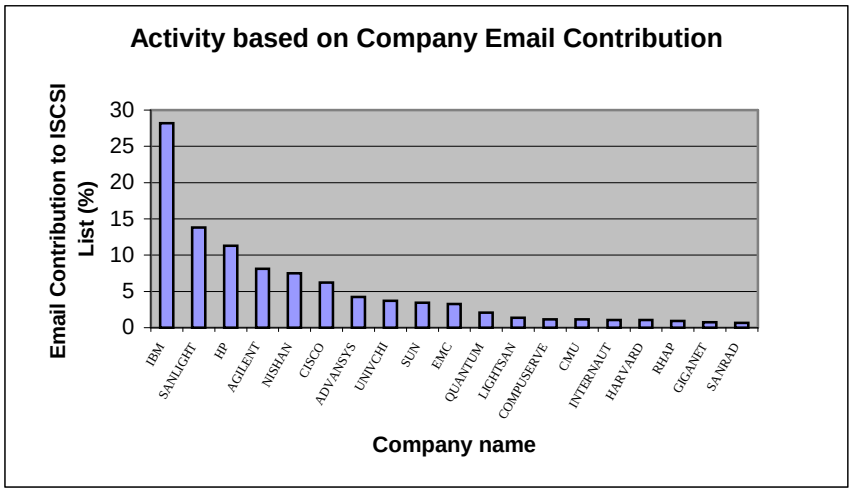


Figure3: The cumulative distribution of contributions based on companies.

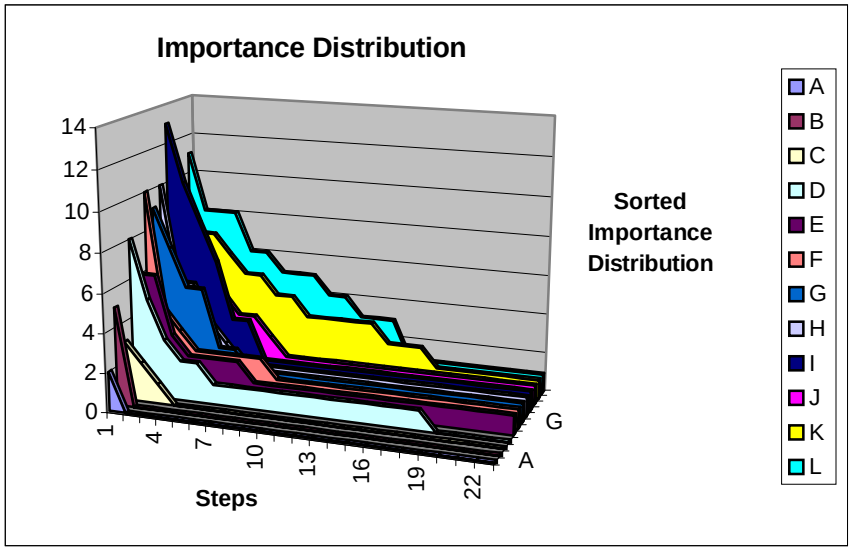


Figure 4: Importance distribution for role analysis. Some people burst responses on specific subjects, while others contribute equal amounts to all discussions sustaining the liveliness of the list.

[3] Note that the x axis is logarithmic in Fig.2.

Fig.4 shows a more dynamic view of the cumulative contributions. *Importance distribution* is the dynamic representation of an individual in the e-mail list showing his/her contributions sorted based on the relative importance the individual gives on some topics. People with different behaviors on the list are taken into consideration. Their cumulative contribution is distributed over the different topics or technical discussions they contributed to and then sorted. Some attendants send e-mails, comments, fixes throughout the lifetime of the project, while some people just burst a number of e-mails over a specific topic and keep silent the rest of the time. The bursts or peaks show the importance the individual gives to specific topics and its distribution shows the overall interest and contribution of the person. Both the shape of the curve and area under the curve have implications about the role of the person. In Fig. 4, the data towards the front is from users who are interested only in one topic. Towards the back the area and thus the contribution increases. These are the people contributing evenly to most of the discussions and sustaining the liveliness of the list.

Once the roles are defined using the cumulative contributions and importance distributions, it may be possible to quantify the productivity of core developers, P(core), and productivity of the community as a whole, P(total). Since ISCSI project develops a protocol specification, we do not have lines of code (LOC) metric at hand. So the following question still remains to be asked and answered: "What replaces the lines of code (LOC) metric from open source code projects in open requirement specification or "open" other projects?"

At this stage we turn our attention to the second community, look investigate whether similar behavior could be observed. With ns analysis we take a different route to answer the same question about the different roles in open source development communities. Another metric that has also been commonly used by previous research is the changes and fixes made to the system and who makes them. This data was collected from the *change history archive* on the ns reflector web page [15]. The information gathered from the *maillist archive* analysis of ns will be used in Section 5.3 to develop metrics related to "customer support in OSS".

The values in Fig. 5 we obtained from the change history. These are not problem or bug reports, but major additions and modifications to the simulator. In total, 35 people made 447 additions and major modifications to the simulator over a period of five years. We see that 50% of these major additions or modifications came from 4 people (half of the pie) and 75% came from 9 people. Independent from the ISCSI project, these numbers from ns project confirm that majority of the changes come from 15-20% of the people in the group.

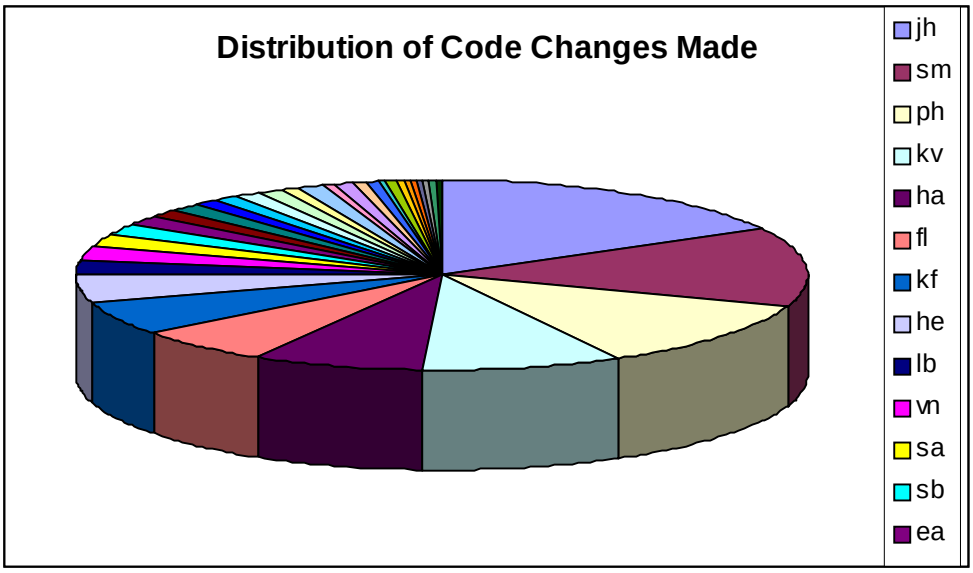


Figure 5: Similar to the e-mail contributions majority of the changes made also come from the core team. Letters on the right are the initials of some of the people that made the changes to the simulator.

5.2 Metrics that relate to the release stability

The next question is to understand the release mechanism, which is a direct measure for the amount and quality of products that are produced by the group. A calculation is given below for ns vs. ISCSI comparison on amount of products:

Project: *Date(end) – Date(begin) / # of releases = release period*

ISCSI: *(3/2001 – 11/1996) = 52 months / 26 releases = 2 months / release*

NS: *(2/2001 – 1/2000) = 12 months / 4 releases = 3months / release*

One could expect on the average 2-3 months to pass between the publications of stable releases in open source developments. The numbers above cannot be metrics by themselves. The question is given these statistics can we develop a metric for *stability* of a release so that we could predict whether it is a good time to publish a new release or not. How do we measure the maturity of our product? Is it possible to come up with a *maturity metric such as 3R(Raw, Ripe, Rotten)*?

Fig. 6 shows the number of changes made per day over all releases. We have counted the intervals in days between releases and recorded the number of changes made to each. Changes/Day metric is a cumulative running average obtained by summing up these intervals and changes:

Changes/Day (R) = (total number of changes made from first release on)/(total days elapsed from first release)

Initially, when there are only few people many releases are published and changes are made on a daily basis. The code is relatively smaller in size at this time. After a while more people start joining and the running average of changes per day gets regulated to 0.33 changes / day, which is around 1 published change/ 3 days.

Figure 7 shows our attempts to derive other stability metrics. We were expecting to see that as code gets bigger the code stabilizes and it becomes harder to make changes to the code. In Fig.7a we see some proof towards our projection. It takes more time to get a new release published, i.e. it is harder to publish something and claim that it is significantly never than the previous one. However, Fig.7b shows that the changes made never stop either. In large projects where new modules are continuously added around to the kernel (core architectural) code, the average number of changes made never stop or stabilize around a fixed value. Although it becomes much harder to come up with new releases, changes made to the code never stabilize. As more pieces are added around the kernel code more changes need to be made to keep the bulk stable. This fact has been quoted as “law of continuing change” governing the development of large programming projects [17] and our results verify the truth of this law. These findings indicate strong similarities between the OSS model and large program development models.

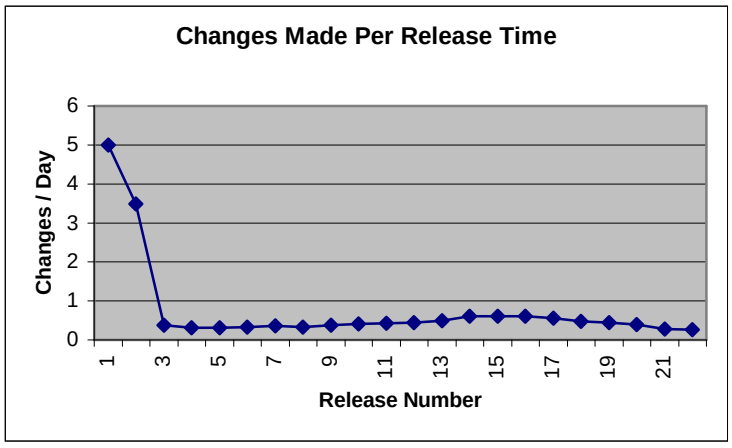
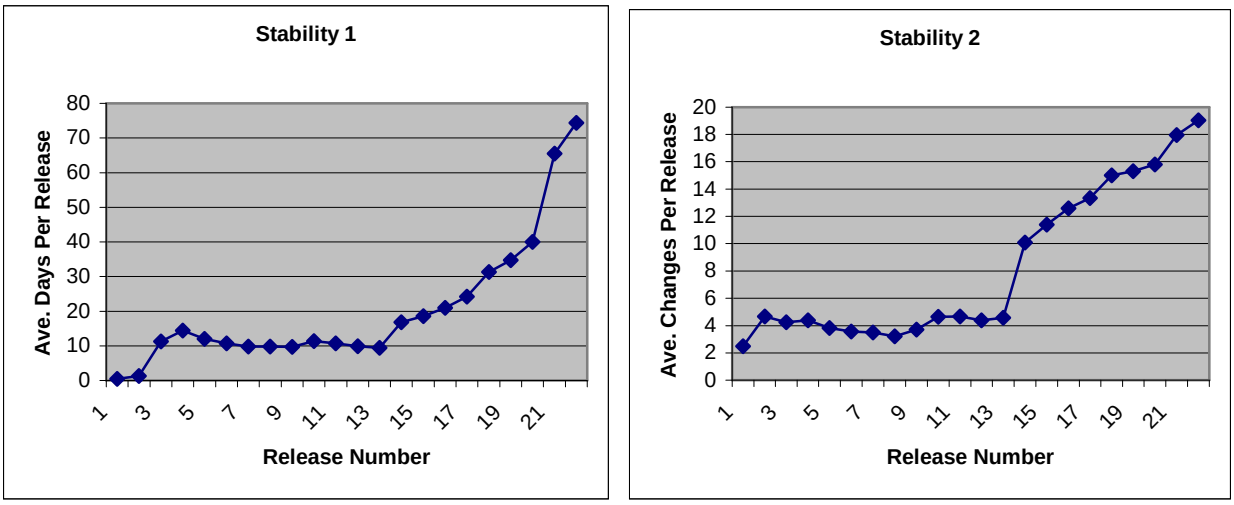


Figure 6: 23 releases were made in the NS project until April, 2001. Summation of changes made over summation of days that elapsed regulate quickly.



(a) (b)

Figure 7: a) First attempt to obtain a stability metric. It becomes harder to make changes, add new functionality and new releases as complexity increases b) Second attempt to obtain a stability metric. NS is a project under continuous development, therefore change rate has not stabilized.

It may look like the metrics we defined for stability analysis have not fulfilled their purpose. However, we must keep in mind that we are looking at ns code that is under continuous development and the metrics clearly illustrate this fact. If the same metrics are used over a project that has a definite functional goal and is finalized at one stage one could expect these figures to reach to a plateau.

5.3 Metrics that relate to customer support

Customer support is an integral part of commercial products, since certain amount of people are always allocated just for this task. The effectiveness and quality of commercial customer support will be kept out of the scope. OSS is a voluntary endeavor, so continuous product and customer support is not guaranteed as it is with their commercial counterparts. To understand what kind of support exists in OSS projects we rigorously parsed through the ns mailing list, which -as of July 2001- is receiving 17 messages on average per day.

Table 2 lists the number of e-mails sent to the ns-list over 4 years. An exponential growth is observed at the beginning. The number started converging over the last few years indicating a natural limit for the number of e-mails that could be handled (without discarding all) by a community of developers. This information brings into mind two other laws quoted for large program development communities by Belady and Lehman [17]. The first is "law of increasing entropy" and the second is "law of statistically smooth growth".

While aiming to gather information about average response time of OSS communities to relate to quality of customer support, we have encountered another interesting behavior. Technical lists have an unseen threshold level, a level that represents the current knowledge and interests of the group. Questions that are too simplistic and that fall well below this threshold may remain unanswered. Questions above the threshold are usually answered and lead to discussions, pushing the general knowledge to a higher level.

Year	Total
1997	411
1998	2096
1999	4144
2000	6352
2001	~7000

Table 2: Number of e-mails sent to the ns mailing list. The value for 2001 has been projected based on the data from first 7 months and growth rate of the list.

Table 3, shows the classification of messages sent to ns mailing list into 9 broad categories. These categories are, technical questions related to networking or the mechanisms of the tool itself; installation, compilation and downloading problems; bug reports; news about upcoming conferences, workshops or other events that may interest the group; patches and scripts written by users that help the analysis of simulation results; generic questions about the list, subscriptions and unsubscriptions; unsolicited messages unrelated to the list or generic comments and finally release information. These categories were generic enough to cover all messages posted and resulted in effective classification.

The rules that have not changed over years (i.e. invariants) are: on average 2 responses are generated to interesting messages that lead to discussions, out of all the questions asked 50% are not given any answer. Technical questions constitute 55-65%, installation problems 15-20% and bug fixes around 10% of the overall traffic, adding up to 80-90% of the total traffic on the list.

Year	TechQ			Install Problems			Bug Reports		
	Ques	Resp	No-res	Ques	Resp	No-res	Ques	Resp	No-res
1997	62	123	47	27	49	13	15	25	3
1998	123	241	131	64	96	22	21	41	22
2000	156	253	164	29	76	17	16	36	15

Year	News	Scripts	List Inquiry	Unsolicited	Release	Total Parsed
1997	6	9	22	4	6	411
1998	0	12	63	2	6	844
2000	3	10	48	8	0	831

Table 3: Classification of e-mails submitted to ns list.

5.4 Similarities between OSS and Large Program Development Model

OSS systems are built by large numbers of volunteers working in a modular, collaborative and massively parallel [8] manner. This fact has been the case with most of the OSS projects that we know of today. It is clear that OSS does not follow waterfall model [18], since there is no strict flow between the design, implementation and maintenance stages. Some stages of Waterfall model may even be skipped in OSS. In the spiral model [19], the life of a project is mapped on a spiral trajectory. The angular dimension represents the progress and the radial dimension the cost of the project. This model is strictly tied to risk analysis of a project and is geared towards minimizing the risks for the success of the development. OSS project communities are composed of volunteers acting without any central authority, which by definition puts these projects at risk at all times.

Dowson [20] classifies waterfall and spiral models as activity-oriented models "which find and execute a plan of actions in a linear fashion" [21]. His classification continues with product-oriented models and decision-oriented models. The process model [21] is an example for decision-oriented models, where the programming activities are defined around four key concepts: "situation, decision, arguments and actions". One deficiency of all these models is that they are either too strict in defining the steps of programming or too abstract to be used as guidelines in real life. Prototyping model emphasizes the importance of quickly implementing a working prototype. This model is an image of "release early, release often" philosophy of OSS, but far from describing the global picture of OSS model. Our findings indicate that more than anything else OSS development model shows the trends similar to and conforms to the metrics listed by Belady and Lehman's "large programming project model" descriptions [17].

This paper lists the "laws of program evolution" as:

1-Law of continuing change: characterized by continuous creation, maintenance and enhancement of the programming systems.

2-Law of increasing entropy: Large projects show an upward trend in the size, complexity and cost. Results confirm nonlinear-possibly exponential-growth and complexity.

Metric	Definition
R	RSN (Release Sequence Numbers)
D_R	Age of the system at release R
$I_R = R_{i+1} - R_i$	Interval between releases
M_R	Size of the system in modules
MH_R	Number of nodules handled (received attention) during interval
$HR_R = MH_R / I_R$	Handle rate (activity produced on release R)
$C_R = MH_R / M_R$	Complexity C_R fraction of the released system modules that were handled during the course of release R.

Table 4: Observable and measurable parameters of the programming process.

3-Law of statistically smooth growth: Software maintenance and enhancement projects are self-regulating organisms. Through questions and answers things become known widespread.

Table 4 list other parameters that were used in their [17] paper to quantify large programming projects.

6 Simulation Model

We need simulations, since e-mail archives can only tell us what has happened after things happen. Without a proper model of the system it may not be possible to project and estimate the behavior of the community and effect of various factors on the productivity and release dates. Simulation enables this sort of projections to be made. With the lessons learned and metrics obtained from the analysis of these two communities we develop a simulation environment that would enable social scenarios to be generated and projections to be made about the productivity of ongoing OSS projects. It is very difficult to model such a complex social system with analytical methods and queuing theory, but a large-scale simulation study of an OSS community may be possible and efficient in producing results.

6.1 Entities in Simulation Model

In our simulation model people correspond to as *network nodes* (hosts), e-mails to *network packets* received by these nodes. These packets (e-mails) are either multicast or *broadcast* to a specific *IP address* (maillist). Fig. 8 illustrates nodes connected by a LAN and receiving packets from the LAN. Nodes have *priority queues* at their network interfaces. Different queuing strategies may be used to represent different behavior. People prefer some messages over the others and in cases of excessive mail traffic they delete their e-mails without reading them as indicated by Table 3 in section 5.3. This behavior is modeled by the *dropping and discarding of packets in the queues.* Bugs are generated using error modules in the simulator and bug fixes will come from nodes as *bug_fix* commands or requests either broadcast or unicast directly to the administrator node.

Local Area Networks (LANs) are the representative bodies, companies, institutions (Fig. 2) under which a group of developers are gathered. LANs connected to a Wide Are Network (WAN), which is the whole mailing list.

E-mail archive and CVS repository (the source code) are controlled by one of these LANs and the nodes in these LANs represent the list administrators and elected project leaders, which are most probably part of the core team. These administrators also answer to the questions with a constant rate consistent with their roles we have found in section 5, Fig.4.

Source code and releases are represented as http files on which requests and modifications can be made. Bug fixes are either broadcast as news packets or http file invalidations.

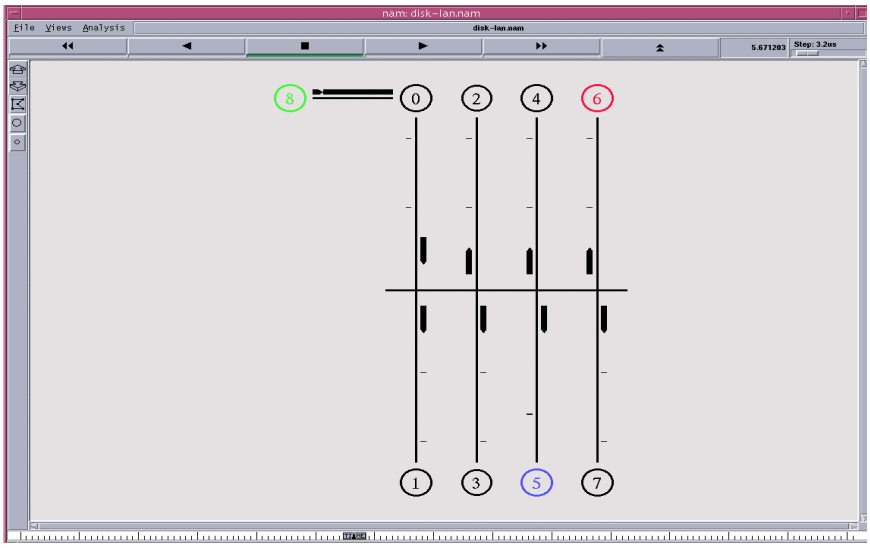


Figure 8: Output of the network animation (nam) tool that comes with ns-allinone package to illustrate and help visualize our simulation model. Nodes receiving packets correspond to people receiving e-mails.

6.2 Modeling scenarios to make projections on success of OSS

After defining the basic entities, it is possible to enhance and fine-tune the model to generate scenarios similar to the ones that occur in real life. These scenarios may lead to valuable questions to be asked and answered about the productivity, customer support, product quality, sustainability of an OSS community. Here we list a few possible scenarios that could be simulated using the network simulation tool.

- Additions and modifications are made to the source code: undocumented changes cause more network traffic than the documented changes and start being generated after a delay.
- Technical questions that are answered (requests receiving responses) increase the global knowledge of the community. Unanswered questions are retransmitted after a timeout and if they go unanswered for a long time, the productivity drops.
- News interesting to the community such as “other company finishing the product” or “conference due date” will be broadcast to cause e-mail traffic storms to see the effect on productivity.
- A active group may be closed in one company? LAN disconnected? What is the effect to the list?
- Addition of new modules to the repository increases the interest to the OSS product, thus more people start joining the list (more nodes are added). This results in more traffic on the list.
- How many people are required to answer? At what rate (percentage) should these people be answering questions and promoting changes?
- Given target productivity levels, time and budget bounds how many people (nodes) and groups (LANs) are required to make this a successful OSS project?
- How are fault penetrations, defects and errors avoided?

7 Conclusions

We tried to contribute to the clarification of OSS development model by making quantitative analysis of two OSS type project. We parsed through e-mail archives and change history files. Our numerical results had parallelism with the results from some previous research. We found that the core team consists of 10-15% of the total number of people on the list and these people make the majority of contribution. We have also come up with other metrics that define the roles taken in OSS communities, the stability of the product releases and response percentages as a measure of customer support.

The lessons learned from the analysis of two communities were used as the basis of a preliminary simulation model that could be used to ask and answer valuable questions to be asked, scenarios to be generated and answered, thus projections and estimation to be made before events take place in OSS development projects.

8 References

[1] Shankland S., Microsoft, Read-Hat argue open source, CNET News.com, July 26, 2001. http://www.new.cnet.com/news/0-1003-200-6690267.html

[2] Microsoft Shared Source, May 2001, http://www.microsoft.com/business/licensing/sharedsource.asp

[3] GNU Public Licence (GPL), http://www.gnu.org/licences/licenses.html

[4] O'Reilly T. Lessons from open-source software development; Communications of the ACM 42, 4 (Apr. 1999), Pages 32 – 37.

[5] Microsoft Shared Source Philosophy: Frequently asked questions, May 17, 2001, http://www.microsoft.com/business/licensing/ssfaq.asp

[6] Johnson K., Open Source Software Development, http://www.cpcs.ucalgary.ca/~johnsonk/seng/seng691/open.htm

[7] Mockus A., Fielding R. T. and Herbsleb J. A case study of open source software development: the Apache server; Proceedings of the 22nd international conference on Software engineering, 2000, Pages 263 - 272

[8] Fitzgerald, B., SEWORLD OSS CFP, Sep/01/2000.

[9] E. Raymond. The Cathedral and the Bazaar. 1998.

[10] MhonArc, An E-mail to HTML Converter, http://www.mhonarc.org

[11] Feller J., Fitzgerald B. A framework analysis of the open source software development paradigm; Proceedings of the twenty first international conference on information systems on Twenty first international conference on information systems, 2000, Pages 58 - 69

[12] Zachman, J. A framework for IS Architecture, IBM Systems Journal (26:3), 1987, pp. 276-292

[13] Satran, J., Smith, D., Meth, K., SCSI over TCP, February 2000

[14] Satran, J., et al. iSCSI (Internet SCSI), IETF draft-satran-isci-01.txt (Jul. 10, 2000); see www.ece.cmu.edu/~ips.

[15] UCB/LBNL/VINT Network Simulator - ns (version 2), http://www.mash.cs.berkeley.edu

[16] Stevens, R. TCP/Illustrated Volume II, Addison Wesley, p-4.

[17] L. Belady and M. Lehman. A model of large program development. IBM Systems Journal, Vol. 15, No.3, 1976, pages 225-252.

[18] W.W.Royce Managing the development of large software systems: concepts and techniques. Proceedings IEEE WestCON, Los Angeles, 1970, pp.1-9

[19] Boehm, B. A spiral model for software development and enhancements" IEEE Computer, 21(5), pp.61-72, 1988.

[20] Dowson, M., Iteration in the Software Process. Proc 9th Int Conf on Software Engineering, Monterey, CA, 1988.

[21] Rolland, C. , Modeling the Requirements Engineering Process, 3rd European-Japanese Seminar in Information Modeling and Knowledge Bases, Budapest, Hungary; 06/1993.