The Open Source Software Ecosystem

Christopher Walton

Division of Informatics
The University of Edinburgh
cdw@dcs.ed.ac.uk

July 10, 2001

Abstract

The Open Source Software (OSS) movement has recently gained much popularity as an alternative method of software development. The basic tenets of OSS development are open source code, free redistribution, and non-restrictive licensing. OSS proponents argue that this approach is faster, more efficient, and leads to a better quality of code. With such successes as Linux and Apache to its name, OSS is now beginning to attract serious attention from the commercial software industry.

The OSS community is a good instance of an information ecosystem. The OSS environment is one which facilitates rapid evolution and change. Consequently, there are many OSS projects with differing behaviours and organisational structures. New OSS projects are constantly being born, and a significant number also fail. In this paper we examine three large OSS projects: Linux, Apache, and Mozilla. Each of these projects displays a different social organisation, and is driven by different forces depending on the niche the product is intended to fill. For each project, we discuss the environment and development process employed, and examine the factors which drive change in this environment. These include both nourishing factors and hazards. We then outline a model of the project as a multi-agent ecosystem which may be used to guide an empirical analysis. Finally, we examine the process by which changes are integrated into the ecosystem and how potential threats are overcome.

1 Introduction

Large software projects must undergo a process of constant evolution, or they risk being overtaken by competing products and new technologies. However, the maintenance of such systems increases in difficulty as the software ages and grows. In recent years, the Open Source Software (OSS) approach to development, previously dismissed as too ad-hoc, appears to provide a solution to many of the problems of software development and scalability. Nonetheless, it is not the case that all OSS projects are developed in the same way, or that they all evolve in the same manner. In this paper we examine three different OSS projects from an ecological perspective, namely Linux, Apache, and Mozilla, treating each project as a separate information ecosystem. We attempt to characterise the environment in which they exist, the forces which drive changes in the environment, and the potential threats to each environment. Our methodology is primarily ethnographic, based on our personal observations and interactions with the open source community.

We begin this paper by presenting a brief history of the OSS movement as a whole, illustrating how it originated and how it came to reach a size worthy of our treatment as an ecosystem. We continue with a definition of the essential notions of informations ecosystems in general. This definition will underpin our models of OSS development, outlined later in the paper. We then limit our discussion to our three chosen OSS projects, presenting a series of our own informal observations of the development environment, driving forces, and hazards of each. As the environments of each of these projects exhibits a different behaviour, we treat each as a separate information ecosystem. We then sketch a formal model for each ecosystem which captures the essential notions uncovered by our informal observations. Finally, we speculate on how changes will be incorporated into each ecosystem and how hazards will be overcome.

The purpose of this paper is twofold; firstly it contains an overview of the development processes of three real OSS projects. This will serve as a useful reference for those interested in understanding the core processes of OSS development. Secondly, the formal models of the environments which we have developed will enable experiments to be performed, and empirical observations to be made. We are currently building a number of tools to perform simulation and analysis of these (and other) models.

2 The OSS Movement

The OSS community may be considered an outgrowth of the underground "hacker culture" which has been in operation since the mid-1960's, originating from such organisations as the now legendary MIT Artificial Intelligence laboratory. In such settings, small groups of like-minded individuals gathered together (often in secret) to work on exciting computer projects. A number of these underground projects developed into commercial entities, e.g. Apple (which stated out as a garage project and developed into a multinational company [Lin99]), many simply vanished, while others continue to flourish as free software projects, e.g. GNU emacs. One of the primary requirements for any OSS project is a large and willing body of like-minded volunteers. Although not the focus of this paper, there are many interesting sociological factors at work in motivating such groups of individuals. These factors are explored in detail in [Ray98], though others such as [Gla00] argue that factors of human nature such as greed may eventually condemn OSS to extinction.

The GNU project was conceived in 1983 as a way of bringing back the cooperative spirit that prevailed in the computing community in early days, and to make cooperation possible once again by removing the obstacles to co-operation imposed by the owners of proprietary software. Led by the outspoken Richard Stallman, the GNU project, and its sister organisation The Free Software Federation (FSF), laid the foundations for the Open Source community. The continuing work of this Free Software community, in producing free tools for software development, has played an essential role in establishing the OSS community. However, unlike the OSS movement, the Free Software community has largely been shunned by the commercial software industry, primarily as a result of its radical political views. There is some degree of overlap between the OSS and Free Software communities, though there is some disagreement on the basic principles [Sta98].

The rapid growth of the Internet, and the advent of the world-wide-web in the mid-1990's suddenly made possible a much larger form of co-operation among previously isolated groups of software developers. As a direct result, a large number of new OSS projects were initiated, and existing OSS projects, such as Linux, experienced both rapid and increasingly globally distributed growth. There are many pragmatic issues to consider in co-ordinating such large groups of distributed developers. Though this is not our primary concern here, these issues are discussed at length in [ČB99]. We now apply the OSS label retroactively to these projects, although this term was not coined until Eric Raymond published a paper entitled "The Cathedral and the Bazaar" [Ray99a]. In this seminal paper he contrasted the carefully architected style of the cathedral with the indisciplined interaction of the bazaar as the defining modes of conventional software development versus the open form of software development which he observed. A large number of advantages of OSS development over traditional closed-source are observed in this paper, for example Linus's Law based on experience with Linux: "Given enough eyeballs, all bugs are shallow". This paper had a profound effect on the community and led to the formation of the Open Source Initiative (OSI) in late 1997. The OSI board consists of a number of prominent figures in the Open Source community including Eric Raymond and Bruce Perens. The OSI is responsible for issuing the "OSI Certified" mark to software projects which conform to the OSS Guidelines [Per97]. The OSI exists to safeguard the interests of the OSS community and to promote the OSS development model as an alternative for commercial software development. The OSI have invested a significant effort into making a business case for OSS development. Eric Raymond attempts to answer the concerns associated with making money from OSS in [Ray99b]. The current success stories for the OSI in commerce include such names as IBM, SGI, Corel, and Netscape. Many such companies are now seriously experimenting with OSS development, or (in the case of Netscape) betting their future on an OSS strategy.

The success of OSS software is currently strongest in niche markets, for example, web servers, programming languages, and embedded systems. This is because the barrier for entry in these areas is low, owing to the lack of entrenched standards, and the rapid pace of development. This is particularly true in the case of the world-wide-web, which has undergone extremely rapid growth, and is one of the areas in which OSS is strongest. OSS is also successful in these back-end areas as the level of technical competence of the users of such software is higher, and therefore there is a greater appreciation of the ability to customise the software at the source-code level. The downside of this is that OSS has traditionally tended to be developed for a technical audience, with user-friendliness and documentation low in the list of priorities. Nonetheless, this is gradually changing as more commercial companies adopt an OSS strategy for their development. It is also worth noting that there is a close connection between the OSS community and academia. In particular, the publication of academic papers closely mirrors the dissemination OSS source code; both operate in a totally open manner in which nothing is hidden, and both are subject to peer review. On a pragmatic level, OSS is also being embraced by academia because of the suitability of open source code for teaching purposes, and the low cost of support.

The area in which OSS software is weakest is the consumer desktop, where Microsoft Windows and related applications are the established standard, and the pace of change is slow. Furthermore, the technical competence of the users of such systems is low, and there is little need or desire for such users to obtain access to the source code. For OSS to succeed in this environment, an OSS product must offer a significant advantage, for example an entirely new *killer application*, or a substantial cost saving over an equivalent closed source offering. There are many obstacles to OSS development in this area including such things as proprietary file formats, licences, and patents. However, OSS is slowly making inroads in this area also, as evidenced by the recent release of OpenOffice by Sun, an OSS competitor to Microsoft Office.

3 Information Ecosystems

The essence of an information ecosystem is a large group of *agents* which exchange *information*. There are many systems which exhibit this behaviour, either in-part or as a whole, though we limit our attention to the OSS community here. There are many different levels at which the OSS community can be considered an information ecosystem. At a very high level, the OSS community as a whole performs as an ecology, with separate OSS projects forming different ecological niches. A large number of OSS projects are connected in some manner, e.g. as part of the same Linux distribution. Therefore, in principle we may produce a loosely-connected model in which each OSS project is modelled by a different agent or group of agents, illustrating the different interactions between OSS projects. However, this kind of model would be very difficult to construct and is somewhat larger than we require for meaningful experimentation. An alternative is a very low-level view of a single OSS project. It is generally the case that OSS projects are structured in a highly modularised fashion. Thus, we may produce a tightly-connected model of the innards of single OSS product, where each agent is a different code module (i.e. a software agent), illustrating how the code itself fits together. However, while this model may be useful as a documentation tool, it will not give the kind of information which we require here. Rather, we will model the OSS community at the level of individual OSS environments.

An OSS *environment* is one in which a large number of self-selected developers participate on a particular project. In keeping with this view, we have chosen to model the environments of a collection of OSS projects, where each is treated as separate ecosystem. It is particularly interesting that the developers in an OSS environment seldom (if ever) meet, rather they co-operate and synchronise electronically, via e-mail, bulletin boards and the web. This is primarily what makes the OSS community an *information* ecosystem. We consider the individual agents in the ecosystem as representing the human developers involved in the project, and the exchange of information to be the the exchange of code between developers. Since we are adopting the same style of representation for each of our chosen OSS projects, a direct comparison between the resulting ecosystems will be possible. This appears to be a good compromise between the high-level view, where project interactions were the main concern, and the low-level view where the program code was the focus of attention.

The central concept of an information ecosystem is that of its *lifecyle*. This has two distinct but interactive meanings in our chosen context. The first is analogous to the biological meaning of the word; the progressive sequence of changes experienced by an organism (i.e. agent) from the moment of fertilisation until its death. For each agent in an information ecosystem, this is determined by the design of the agent, and the environment through which it interacts with the other agents under resource constraints. The second meaning of "lifecycle" is the design lifecycle for a software system. This relates to the code upon which the agents will operate. A suitable interaction between the lifecycle of the agents and the lifecycles of the code upon which they operate will ultimately determine the fate of the ecosystem as a whole.

We will now consider our three chosen OSS projects in-turn. It is interesting to note that even though they are all relatively large and successful, the environments in which they exist, and the driving forces behind each differ considerably. This will be reflected in our ecological models. Later in the paper we perform a comparison between our ecosystems in an attempt to describe which is "best" in terms of such properties as adaptability and sustainability.

4 The Linux Environment

Linux is an OSS project which provides an implementation of a UNIX-like operating system kernel. Much has been written about the history of Linux, e.g. [DO99], and therefore we will not repeat it here. Linux is currently the only general-purpose operating system with an increasing market share. We only consider the Linux *kernel*, which is the common element of the many Linux *distributions*, e.g. RedHat, Debian, SuSE. These distributions package the Linux kernel with a large amount of other OSS software in order to make a useful operating system product.

Linux is one of the largest, longest-lived, and successful OSS projects. It is therefore surprising that the environment in which Linux is developed is probably one of the most primitive of any OSS project. The Linux kernel is principally controlled by Linus Torvalds who was responsible for the initial code. Linus maintains absolute control over the official Linux code-base, with the final word on which changes are accepted. This is often referred to colloquially as a "benevolent dictatorship". Linus maintains the view that this method of operation ensures a high quality of code and a fast pace of development. The argument being that a committee of developers would often be forced to reach a compromise, or be stalled by internal disagreements. Although there have been several high-profile disagreements among developers, it is generally agreed that this method of working is successful, since Linus' only motivating factor is to improve the code. Therefore, anyone who can make a sufficiently convincing argument for their proposed change, and produce code of an acceptable quality, is likely to have it accepted. It is worth noting that licensing of the Linux code is such that anyone could fork off their own version of Linux, though this is strongly discouraged in the Linux community.

There are no code revision tools (e.g. CVS) currently used in the development of the Linux kernel. Instead, there are generally two versions of the code-base active at any time; the *stable* branch, and the *development* branch. The stable branch is intended for production use and the only changes permitted to this code are bug-fixes or features which do not affect any other code. The development branch is the focus of Linux development where anything may potentially be changed. Periodically, the development branch undergoes what is termed a *feature freeze* where the developers stop adding new changes and concentrate on fixing the bugs. After a sufficient period, the development branch becomes the new stable branch, and a new development branch begins. Changes to the Linux code are submitted in the form of *patches* to the public `linux-kernel` mailing list, or directly by e-mail to Linus. The actual Linux code base is maintained privately by Linus, who releases periodic public snapshots on the `kernel.org` web site. The majority of communication between developers also happens publicly on the kernel mailing list, although there are a number of other public mailing lists for discussing more specialised areas of the Linux kernel. Discussions on the mailing list often result in private e-mail communication between developers, when there are disagreements or detailed technical matters to discuss, though the results of such discussions are usually summarised on the public mailing lists to keep other developers informed of any decisions. The development of Linux is therefore completely open; anyone may read the mailing lists, download the source code, or submit patches against the code-base.

The revision process outlined above has been in place since the early days of Linux development and has proved successful. However, the reliance on Linus alone to approve changes to the code and to synchronise the submission of patches has occasionally lead to bottlenecks in the development process. Consequently, the development environment has been forced to undergo a number of changes to alleviate this problem. It is now the case that simply posting a patch to the mailing lists or directly to Linus is unlikely to prove successful. Instead, the kernel code has over time become loosely divided into a number of sections, each of which is maintained by a separate developer or group of developers. Although there is no real concept of ownership of sections of the code, is usually the case that a developer who has been working on a particular part of the code for a long time assumes responsibility for it. These developers are often referred to as *top-level* developers. The selection of the developers is essentially an informal meritocracy. These developers synchronise patches to their sections of the code, and feed the results to Linus, thereby relieving Linus from a significant proportion of the burden. This was recently illustrated by the delegation of the entire stable branch to Alan Cox, who collects kernel patches against the stable kernel, only seeking Linus' approval for the final result. When competing solutions to a particular problem arise, or when there is a disagreement among the top-level developers, a decision by Linus is usually required.

Over time, the depth of the developer hierarchy has increased, such that each section in the kernel is divided into subsections which are maintained by sub-developers who communicate only with developers above them in the hierarchy. When the number of developers in a particular subsection reaches a reasonable level, a separate mailing list is created to reduce the load on the main kernel list. This is illustrated best by the presence of ports to other architectures in the Linux code. The main kernel development is done on the Intel architecture, which is the most widely used platform. Ports of Linux to other architectures (e.g. Alpha, Sparc) are controlled by top-level developers who maintain a separate mailing list and code-base against which other developers submit patches. These top-level developers then periodically synchronise their separate code-base with the main Linux code controlled by Linus.

4.1 Environmental Forces

The pace of development of Linux is extremely rapid. New snapshots of the development kernel code are released at approximately weekly intervals. There are many driving forces behind the project which are responsible for this behaviour. One of the main driving forces is a great desire in the Linux community that the kernel must exceed, or at least attain feature parity with the main competition, namely the Microsoft Windows operating systems. This was illustrated by the Mindcraft benchmarks [Keg99] which showed that Windows NT would outperform Linux under certain extreme circumstances. The development community immediately responded, resulting in a flurry of activity to overcome the bottlenecks which were highlighted by the benchmark.

The link between Linux and the OSS community is another significant driving force in the environment. Linux is a leading exemplar of a successful OSS project. Furthermore, Linux is the platform on which a great many other OSS projects are developed. Therefore, as the OSS community grows, it is likely that the Linux community will grow in tandem. An empirical study of the size of the Linux kernel over time is presented in [GT00].

Linux is now perceived by the computing community at large as the main rival to the Windows operating systems. Consequently it has attracted the attention of a number of commercial companies which are in competition with Microsoft. These include names such as SGI, Sun, IBM, and Corel. These companies are now actively participating in the development of the Linux kernel and/or associated applications. Microsoft's recent legal troubles, such as the Department of Justice (DOJ) investigation, have highlighted the fact to many companies that the Windows monopoly may not continue indefinitely. As a result, a number of them have now begun to explore Linux as an alternative. The OSS nature of Linux is particularly attractive to these companies, since they can customise, develop, and release their own versions of the code. This also acts as a form of guarantee; that if the Linux project should collapse, they could continue to develop and release their own code. A large number of new companies have also been formed around Linux. These include companies which produce packaged distributions for Linux, others which provide support for Linux users, others which develop applications for Linux, and other which design, manufacture, and assemble hardware for the Linux platform. All of these companies bring their own development pressures to the Linux community, further increasing the pace of development. In a number of cases, these companies are directly funding Linux kernel developers to ensure that their needs are being represented in the Linux community.

A great deal of momentum has also built up in the Linux community, owing to its increasing penetration into new areas such as embedded systems and mainframes. This in turn has led to an increase in development activity as new challenges in these domains arise and are overcome. The result is that there are versions of Linux available from pocket computers, e.g. the Compaq iPaq, right up to mainframes e.g. the IBM S/390. The impressive scalability demonstrated by the Linux kernel is primarily as a result of a desire within the development community to do things the best possible way, with an absolute intolerance of kludge or bloat.

This is enforced by Linus who refuses to allow any changes to the kernel which violate these principles. The kernel code is in a constant state of revision. Many parts of the code, e.g. the networking layers, have been rewritten multiple times in an attempt to make them leaner and faster. Unlike other operating systems, the kernel APIs in Linux are constantly changing as the code is modified. Although this will break any code which is dependent on kernel-level structures, the availability of the source code for these applications means that most problems can be overcome by a simple recompilation step. This neatly avoids the accumulation of debris associated with backward-compatibility layers for supporting legacy closed-source binary applications.

4.2 Environmental Hazards

Despite the success of Linux, there are a number of potential hazards which may inhibit the future growth potential of the environment. The main threat appears to come from the lack of coordination within the community. Although we have described how the Linux environment has effectively organised itself into scalable hierarchy, there are nonetheless a number of potential problems with such a self-organising structure. Currently, developers work on a part of the kernel which interests them the most, with the result that the less interesting tasks, such as documentation, get left until last or remain incomplete. As the size of the code increases, it is likely that the number of such holes will increase. An example of such a hole occurred recently with the implementation of the Network File System (NFS) in the Linux kernel. For a long time, this part of the kernel was not developed, with the result that the NFS implementation in the kernel has lagged behind that of other UNIX operating systems. Another serious problem with the current hierarchy is that of developer *starvation*. As the depth of the developer hierarchy increases, there will be new developers who are willing to contribute to the code-base, but are too far down the order to work on problems that they are interested in, eventually leading to the developer losing interest and abandoning the project.

The Linux developer community has adapted over time towards a scalable hierarchy. However, it is not certain that this hierarchy can grow indefinitely. A new developer who wishes to begin kernel development must find out who the relevant maintainers are for the code that they are interested in modifying, then canvass the opinion of these developers, and then produce a good solution to the problem. The increasing complexity of the code, and the increasing depth of the developer hierarchy, has led to some recent frustration regarding the relatively high barriers for entry imposed on new developers. Often new developers find that they are simply ignored, though this is generally because they have failed to follow the necessary coding practices, or failed to take into account other existing development work. A number of developers are of the opinion that a high entry barrier is a good thing because it helps to maintain a high level of code quality. Nonetheless, it is the case that developers frequently change their interests over time or leave the project, leaving room for new developers to take their place.

The recent influx of commercial interest in the Linux kernel may also have a negative impact. At present, there is little commercial interference in the development process. However, it is likely that there will arise a situation where commercial pressures for a particular kernel feature severely disrupt the development process. This has been illustrated to some extent with the recent lobbying on the kernel mailing list over the inclusion of the ReiserFS journalled file-system in the kernel. Another problem with commercial involvement is that of patenting and licensing issues. There is a clear conflict between the commercial desire to protect intellectual property from competitors, and the desire to maximise support for a particular product. As Linux increases in popularity, this may eventually lead to a situation where Linux development is halted as it cannot support a particular essential device due to patenting or licensing restrictions. However, where this has happened in the past, the Linux community has successfully persuaded the company in question to provide support, or discovered a suitable work-around.

A real hazard in the Linux community is posed by over enthusiasm. Due to the desire by developers to constantly improve Linux, it is very difficult for the project to stabilise. Developers are often more interested in adding new features to the project, rather than concentrating on the tedious task of fixing all the bugs. Also, it is often the case that the fix to a bug involves a significant reworking of a part of the kernel, introducing more bugs in the process. This has led to the undesirable trend of long development cycles between stable releases of the kernel. The provision of stable releases is essential for the success of Linux, since these releases are required for real production use. However, despite the announced intention of Linus to produce frequent stable releases of the kernel, this has not happened. The current development branch (version 2.3) of the kernel has been in a state of flux for nearly two years. It is a tradition in the OSS community that deadlines should not be set for software releases. However, in future it is essential that these development cycles are reduced in length, though this must be done without dampening the enthusiasm of the community.

4.3 A Linux Ecosystem

We have now outlined in detail our observations on how the Linux development environment operates. However, in order to reason at a quantifiable level about the effect of changes to the Linux environment on the development process, we require a more formal model which abstracts away unnecessary detail. What follows is a proposal for a model of the Linux environment as an information ecosystem.

Our information ecosystem is composed of a large number of individual agents. Each *agent* in this ecosystem is representative of a human developer in the Linux community. We use the term *ecosystem* to reflect the fact that there are very large numbers of agents involved. Our ecosystem is structured via *message-passing* between individual agents. We are not particularly concerned with the medium by which these messages are passed, e.g. e-mail or web, only that messages are being exchanged. We consider agents to possess *resources*, which we equate with code. Messages are responsible for transferring resources between agents, i.e. messages are equivalent to code patches. The ecosystem is highly dynamic in that agents are constantly joining or leaving the ecosystem.

From our previous discussion, it is apparent that there are distinctions between the agents in terms of roles and responsibilities of the developers. We have isolated what we believe to be four different types of agents in the Linux environment. The different kinds of agents, and an approximate structure of the ecosystem is illustrated in Figure 1. Note that the roles of the agents will change during the lifecyle of the ecosystem, i.e. the topology is not fixed.

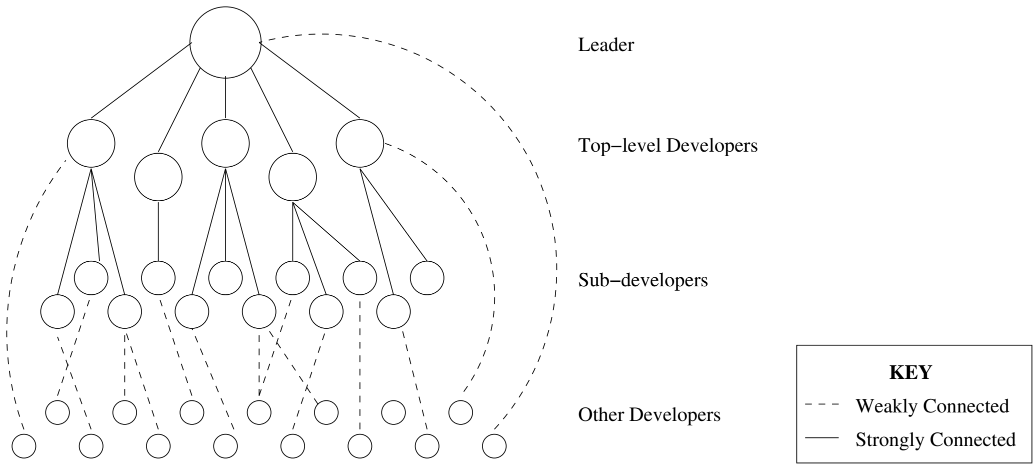

Figure 1: The Linux Ecosystem

The first kind of agent is the project leader, representing Linus Torvalds. There is only one such agent in the ecosystem, and this agent has control over the central resource: the Linux code-base. The amount of resource (i.e. code) for this agent will increase over time as messages are received from other agents. To represent arbitration, the agent will choose to *accept* or *reject* messages. This is very similar to the current submission of patches to Linus: patches are generally accepted and rejected from the Linux code-base without explanation. It is our intention to introduce *negotiation* into the model at a later stage. However, a simple accept/reject mechanism is enough to represent the underlying behaviour.

The second kind of agent represents the top-level developers, e.g. Alan Cox. This kind of agent is responsible for receiving messages from other agents, and for sending messages directly to the leader agent. The amount of resource for each agent will vary over time; increasing as messages are received and accepted, and decreasing as the accumulated resource is sent and accepted by the leader. The number of these agents will be small, and will only change gradually over time.

The third kind of agent is representative of a sub-developer. These are developers who are closely linked to the development of the Linux kernel, but are not top-level developers. This kind of agent will generally co-operate closely with a single top-level developer. The level of resource for this agent will typically be small. The level will increase as the agent produces code, and then deplete to zero as the code is accepted by a top-level developer. This kind of agent will seldom receive messages itself. There will typically be a large number of such agents in the ecosystem, and the level will vary significantly over time. A number of sub-developers will eventually become top-level developers, but this will only occur infrequently.

The final kind of agent is a general category representing all other kinds of developers. These are developers who are only loosely associated with the development process, e.g. a developer who submits a bug fix. These agents will possess only very small quantities of resource, if any. There will typically be very large number of such agents, and these agents will not be closely associated with any one top-level developer or sub-developer, and therefore exist outside the main strongly-connected hierarchy. The number of these agents will vary to a great extent over time. A proportion of these agents will become sub-developers, while many others will leave the ecosystem or become dormant. The level of activity for a particular agent can be measured by considering the rate of accepted resource that they produce over time. Developers with a high level of activity will be promoted in the hierarchy.

Our model is defined such that the only agents which may join the ecosystem are in the general developer category, i.e. it is not possible to create top-level developer or sub-developer agents directly. However, any agent (including the leader) may potentially leave the ecosystem. It will usually be the case that an agent which leaves the ecosystem will immediately be replaced by an agent in the category below. It is clearly the case that as the ecosystem grows in size, it will become more difficult for an agent to become a top-level developer or sub-developer. This is a very close approximation of the behaviour which we have observed in the Linux community.

It is our intention to explicitly model the Linux ecosystem using the simulation tools which we have developed [Vas01]. This model will enable us to examine the lifecycles of the ecosystem as it grows and adapts to changes. As an example, we can simulate the effect of the leader agent leaving the ecosystem, and examine how the ecosystem adapts to meet this change. This would directly address one of the most common fears in the Linux community that the project will collapse if Linus Torvalds were ever to leave. Other properties, such as the growth of the hierarchy and developer starvation can be explored in a similar manner. If we focus on the levels of resource in the ecosystem, we can explore hazards such as stale code, bloat, code forks, and the effects of long development cycles. It is our hope that we can demonstrate the sustainability of the ecosystem in the presence of such hazards, and thereby further validate the OSS development model.

5 The Apache Environment

Apache has been the most widely deployed web server on the Internet since 1996, currently used by over 60% of sites worldwide [Net00]. The Apache Server project is an effort to develop and maintain an OSS web server for a variety of operating systems, including Windows and Linux. We will only consider the Apache Server here, rather than the full list of projects operated under the Apache Software Foundation umbrella. The success and growth of Apache closely parallels the Linux operating system. However, it is interesting to note that the organisation of the environment, and the driving forces behind this environment are considerably different from that of Linux.

The Apache Group (AG) began in 1995 as a group of 8 voluntary core developers who coordinated changes (by means of patches) to the NCSA `httpd` web-server. A year later, the AG had completely rewritten the NCSA code-base, and Apache had become the most popular web server on the Internet. A brief history of the Apache project is described in [Gro99]. The Apache group currently consists of 25 core developers and is still by far the most popular web server. An empirical study of the growth in size of the AG and the code-base appears in [MFH00].

The Apache development environment has a considerably clearer and more rigid structure than the Linux environment described previously. There is no central figure in the AG equivalent to Linus Torvalds. Instead, the Apache code-base is controlled by a group of core developers who make up the AG. The role of these core developers, and the selection of new core developers is described in [Fie99]. Each of the core developers are responsible for a separate part (or parts) of the code. The core developers coordinate the submission of patches to the code under their control. While Linux operates on a "benevolent dictatorship" model, the Apache group is much closer to a democracy; all major changes to the code-base are voted on by the core developers. The selection of the core developers is also performed democratically. A developer who has contributed to the project for a significant period of time, usually 6 months, can be nominated for membership of the core group. The other core developers then vote on whether to accept the developer into the group. Several of the core developers have additional roles such as vote-coordinator or release manager, though these roles are periodically rotated.

Over time, the organisational structure of Apache has grown to resemble something akin to a conventional business. The AG is now a subset of the Apache Software Foundation (ASF), founded in 1999. The ASF oversees a group of projects including the Apache server. The role of the ASF is in providing organisational, legal, and financial support for its projects. The ASF operates as a non-profit organisation with an elected board of directors, and has a set of corporation byelaws. Nonetheless, despite the large organisational differences between Linux and Apache, the high-level view of their environments is largely similar. Although there is no central figure in control of the Apache Group, it is the case that before a major release, a developer assumes the role of release manager to ensure the quality of the code. Similarly, both projects consist of a large group of volunteer developers, structured as a hierarchy which is determined by a form of meritocracy. The main difference is that this structure arose naturally within Linux, while in the Apache Group it was dictated by the project guidelines.

The code revision process in Apache is similarly more rigidly controlled than in Linux. As with Linux, there are generally two versions of the code-base in development at any time; corresponding to a stable version for production use, and a development version. The code-base is maintained by a public concurrent versions system (CVS) database. The use of CVS allows a great deal of flexibility concerning the application of patches to the code-base, e.g. patches to the code can be reverted at any time. Only the core developers have write access to the CVS database, though anyone can have read access. For convenience, snapshots of the CVS code are made periodically, corresponding to different release numbers. The submission of patches to the code, and the discussions among developers, occurs via a number of mailing lists and newsgroups, and a development web-site. It is worth noting that unlike Linux, some of these developer resources are private, e.g. for dealing with security issues. There is also a GNATS web-based bug tracking system which enables bug reports to be easily submitted and tracked. It is also worth noting that some of these communication channels to the developers are better than others. In particular, the newsgroups have a particularly high level of noise and are therefore the least likely means to attract the attention of the developers.

5.1 Environmental Forces

The motivating factors and the driving forces behind the Apache project are very similar to those which we have identified in Linux. Both operate in a rapidly changing environment with stiff competition. Just as the main competition to the Linux is the Microsoft Windows operating systems, the main competition to Apache is the Microsoft Internet Information Servers (IIS). The difference being that Apache is in the dominant position, while Linux is still a minority operating system. This is partly as a result of the fact that Apache works in conjunction with Windows, rather than being a direct competition. However, the link with the OSS community and Linux is undoubtedly in itself a motivating factor. Both Linux and Apache have a very strong team of highly-motivated developers. In both groups, there is a strong desire to produce the best possible product; the desire for high performance and avoidance of bloat are just as strong in the Apache group as they are in Linux.

One of the criticisms that is often levelled at OSS projects is that they have no explicit system design. This was initially true of both Linux and Apache projects. It is therefore interesting to note that both Linux and Apache have evolved into highly modularised and extensible products which can rapidly adapt to new technologies. Linux has developed a sophisticated modules system for device drivers, enabling the kernel to be customised, and new devices to be easily supported. Apache has similarly developed a sophisticated modules architecture which enables new protocols to be easily supported e.g. SSL, Perl, PHP. This level of modularisation is made possible only by the OSS nature of the product and is unrivalled by competing closed-source products.

The popularity of the Apache server has attracted considerable commercial interest, just as it has done for Linux. There are many companies which are loosely associated with the Apache server, e.g. Linux vendors who bundle Apache with their distributions. There are also a number of companies which produce commercial solutions based on Apache, and enhanced with proprietary extensions, e.g. C2NET Stronghold, and Covalent products. However, the company which attracted the most attention when it became closely associated with Apache was IBM. In 1998, IBM entered into a partnership with the Apache Group, whereby IBM would supply technology and manpower to the Apache project, while using the Apache server as the core of their WebSphere application software. It is interesting to note that this partnership arose not because Apache needed the help of IBM, but rather because IBM's own closed-source web server products, e.g Lotus Domino Go, were failing to compete with the OSS Apache product. IBM also chose to work within the existing AG framework, rather than developing their own version of the software. The partnership between Apache and IBM has so far proved to be a successful driving force in the project. IBM has made a number of significant contributions, particularly in the area of support for Windows NT where Apache was previously lacking, and IBM had gained significant benefit itself from being associated with the Apache project [McK99].

5.2 Environmental Hazards

Despite the success of the Apache project, there are nonetheless a number of hazards apparent in the Apache community. As with Linux, the Apache project suffers from the problem of developer over-enthusiasm. The constant desire to improve the Apache server has led to increasingly long development cycles between production releases. As a result, there is an increasing frustration in the community over the delay in releasing the much anticipated 2.0 version of Apache. Frequent production releases of the code-base are essential, as these releases are required for real-world use. It is usually the case that there is a significant delay between a production release, and its adoption by the community at large. Consequently, as the delay between production releases increases, there is the danger that the popularity of Apache may diminish as it is perceived as being out of date.

Unlike the Linux project, which may suffer in the long term through disorganisation in the environment, the Apache project may suffer from over-organisation. As the project grows in size, it will become more difficult to obtain a consensus between developers on certain decisions. At present, changes are only accepted if there are no negative votes. This will likely have to change in future, or the project will stall. However, at present this approach has proved important in ensuring the quality of the code. The rigid organisation of the project may also make it difficult to perform wide ranging changes on the project. In Linux, where there is no rigid ownership of the code, large changes which substantially change the code-base are possible, and indeed have occurred. However, in Apache, where each module is maintained by a specific developer, it is difficult to imagine that a substantial restructuring would be possible without imposing a significant strain on the organisation. Nonetheless, one could also argue that such large changes are undesirable, and that enforcing a process of small incremental changes on an already successful project will ensure future success.

5.3 An Apache Ecosystem

Our ecological model of the Apache environment is illustrated in Figure 2. As in the Linux ecosystem, individual agents refer to human developers, and agents possess resources which correspond to code. However, owing to the fact that Apache operates as a democracy, with the concept of voting central to the operation of the ecosystem, we introduce a new message type to represent a *vote*; either an acceptance, a rejection (veto), or an abstention. This is in addition to messages for transferring resource between agents (i.e. code patches). We have isolated only two different kinds of agents in the Apache ecosystem; *core-developers* and *general developers*. However, it is worth noting that the core-developers may have a number of additional responsibilities, which we describe below.

There is no central agent in the Apache environment, thus the resource (i.e. code-base) is distributed between the core developers. The level of resource for the core-developers will therefore remain relatively stable over time, increasing as new code is produced or accepted from external sources. During release cycles, a core-developer agent will assume the role of *release co-ordinator*. This agent will collect the resources from

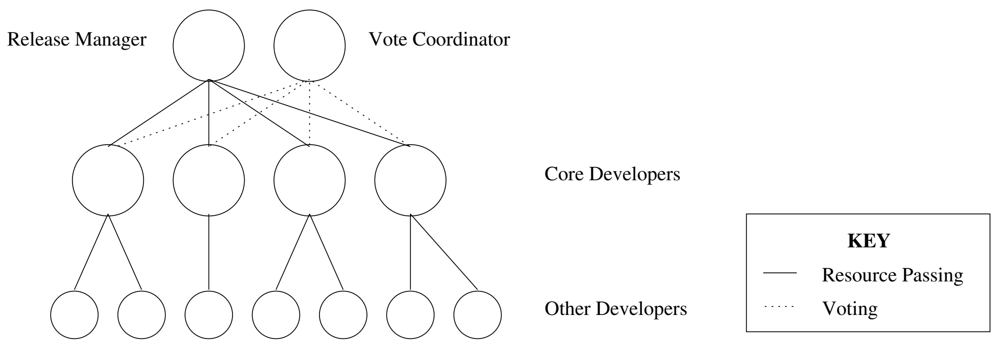

Figure 2: The Apache Ecosystem

the other core-developer agents. The role of release co-ordinator will rotate between the core-developers. The core-developer agents do not decide themselves whether to accept or reject code. Rather, a vote involving all the core-developers must be taken. To facilitate the voting process, one of the core-developer agents will assume the responsibility of *voting co-ordinator*, responsible for collecting the voting messages and issuing the final verdict. The voting process will be subject to the identical rules of the Apache environment, i.e. three acceptance votes, and no rejection votes are necessary for accepting a patch. The role of voting co-ordinator will rotate among the core-developers over time.

The general developer agents are usually associated with a particular core-developer. This category of agents consists of developers who are both closely and loosely associated with the development process. These agents will accumulate small quantities of resource and then submit it to a particular core-developer for consideration. The number of agents in this category will change significantly over the lifecycle of the ecosystem, unlike the number of core-developers which will remain relatively stable.

General developer agents do not participate in the voting process, and therefore exhibit simpler behaviours than the core-developers. General developer agents will usually be considered for promotion to core-developers when they have been active for a significant period of time. The activity of an agent can be quantified by the level of resource which it produces. The acceptance of a new core-developer is performed by a vote among the existing core-developers. Many of the general agents will also leave the ecosystem or become dormant.

We only permit the creation of new general developer agents in our ecosystem. Core-developer agents must be created through the voting process. Consequently, the ecosystem will initially contain a number of core-developers to enable progress. This corresponds closely to the initial "gang of eight" developers who began the Apache project. Core-developers are permitted to leave the ecosystem, at which point their resources will be transferred to another core-developer.

The Apache model of the ecosystem is clearly rather different from the Linux model. By simulating these ecosystems with a similar set of initial conditions, we can compare the effectiveness of the two in terms of growth and response to hazards. It is possible that the shallow hierarchy of the Apache project will prove to be a bottleneck as the ecosystem grows in size. Consequently, it is our intention to examine the effect of introducing a layer of sub-developers, as in the Linux environment. These developers may have a separate system of voting, or participate in the main voting system with a lesser weight allocated to their votes. It may also be the case that the strict voting system may serve to stifle progress of a larger ecosystem, and therefore we will also examine the response of the ecosystem to a range of voting strategies. Furthermore, it will be interesting to explore the response of the ecosystem to the introduction of hazards. In particular, the Linux style of development is often cited to be an advantage over a democratic system as it allows a rapid response to changes and threats. Our models will allow quantifiable experiments to be performed over the lifecycles of the ecosystems, with the intention of establishing whether the "benevolent dictatorship" or democratic system is more appropriate for OSS development.

6 The Mozilla Environment

The Mozilla project is an attempt to create an open source web browser, designed for standards compliance, performance and portability. The Mozilla project began in 1998 when Netscape Communications Inc. made the decision to transfer the development of their Communicator product from an internal closed-source process to a public OSS process. This was hailed as a great success at the time for the fledgling OSI. However, despite initial optimism, the project has not turned out to be as successful as was initially anticipated. In particlar, after nearly three years of development, there has still been no stable release of the code-base. Other factors, such as the dwindling market share of Netscape and its subsequent acquisition by AOL, have meant that the future of the project has been under continual doubt. Nonetheless, there are signs that the Mozilla project may yet achieve its goals, which we will discuss later.

The Mozilla project began at a time when both Linux and Apache were well established OSS projects. It is apparent that the founders of the Mozilla project attempted to combine the best aspects of both of these projects. Consequently, the Mozilla development environment is a curious amalgamation of both (contrasting) styles. At a high-level, the Mozilla Project is co-ordinated by the Mozilla Organisation (MO). The MO is not responsible for producing the actual Mozilla code. Rather, it exists to provide direction for the project, to provide financial and legal support, and to co-ordinate the communication channels between developers. The MO is mainly comprised of employees of Netscape (now AOL). The MO can be viewed as having a similar, though more hands-on, role to the Apache Software Foundation. The actual Mozilla code is divided into modules in a similar style to Apache. Each module has an explicit *owner* (also called the *primary coder*) who is responsible for co-ordinating bug fixes and changes to the module. Many of these module owners are also Netscape employees, though some are not. A module also has a number of *peers*, who contribute to the code of the module through the module owner. These peers are generally volunteer developers from outside Netscape. Thus, the explicit structuring and ownership of the Mozilla code is very similar in style to the rigid hierarchy of Apache. Nonetheless, the Mozilla project is not operated on the democratic system employed by Apache. Instead, Mozilla has adopted the "benevolent dictatorship" approach of Linux. The MO has the additional role of deciding which changes to the code are actually accepted into a particular release of the code-base. In this respect, the MO performs the role of the leader in the Linux environment. The owner of each module also operates as a benevolent dictator, who decides which changes are accepted into a particular module. Nonetheless, it is possible for another developer (or group of developers) to produce alternative code for a particular module and have these changes accepted by the MO, usurping the module owner. This is very similar to the way in which patches in Linux can be submitted directly to Linus, bypassing the core developers. By combining the organisational structure of Apache with the flexibility of Linux, it is clear that the intention was to capture the best aspects of both projects while avoiding a number of the hazards which we highlighted earlier, i.e. poor scalability and inflexibility.

Code revision in Mozilla is tightly controlled through a publicly accessible CVS database. Only the module owners have write access to the database, though anyone can examine the CVS tree to determine the state of the project. This is very similar in style to the code revision in Apache. Nightly builds of the CVS tree are also made available for testing purposes, and periodic "Milestone" releases are made when certain goals in the project are met. At present, there is only one active version, but it is likely that the code will divide into stable and development branches upon the first release on Mozilla. It is also worth noting that a closed-source revision of the Mozilla code is also maintained internally by Netscape/AOL for their commercial "Netscape 6" product. This is an extension of Mozilla with a number of proprietary modules, e.g. DOM scripting. Netscape/AOL is therefore in the unusual position of maintaining OSS and closed-source versions of the same code base. The fact that the OSS Mozilla code is responsible for feeding the closed-source Netscape product has irritated some members of the OSS community. However, the situation is not that different from the commercial distributions of Linux and the proprietary versions of Apache.

Bug tracking in Mozilla is handled by a sophisticated web-based bug-tracking database system called Bugzilla. Bugzilla is itself an OSS project maintained by the Mozilla Organisation. The Mozilla approach to bug management is far more advanced than the GNATS-based bug tracking system used in Apache. In [MFH00] it was noted that the Apache bug database is rarely used by the developers, rather the mailing lists and newsgroups are the primary methods of bug reporting. By contrast, Bugzilla is a key component of the development of Mozilla. The use of Bugzilla in the Mozilla project allows the lifecycle of each bug to be tracked through the system. Bugzilla has many sophisticated features, such as automatic bug reporting, and the ability to track dependencies between bugs. Bugzilla also tracks requested features (RFEs) in addition to genuine bugs in the system. A Quality Assurance (QA) team has also been created to assist with reporting bugs, and ensuring that closed bugs have been adequately fixed. Nonetheless, despite the use of such strict bug controls, it is generally acknowledged that Mozilla is infested with bugs, far more so than either Linux or Apache. However, this may be a result of its Netscape Communicator heritage, rather than a result of the bug management system. Though it could be argued that a laborious bug management process may actually dissuade developers from fixing bugs.

6.1 Environmental Hazards

The original intention of the Mozilla Organisation was that a large volunteer effort would drive the core programming of the browser. However, the uptake of development from outside Netscape/AOL has been very slow. Currently, the number of internal Netscape developers far outnumber the external volunteer developers. The reasons behind this problem are discussed in [Ray99a, Hec99, Zaw99]. The initial slow uptake of the project by external developers was almost certainly due to the "spaghetti" nature of the code-base, derived from the commercial Netscape Communicator product. This is a very different situation from the Linux and Apache projects which were developed as OSS from the beginning. Unfortunately, many features of the Communicator software were proprietary in nature and were removed prior to the Mozilla release. As a result, the initial release of the code was very much incomplete and required that developers obtain a licence for the proprietary Motif library. Therefore, new developers could not easily build or run the initial software. The problem was further compounded by the complexity of the code, being a mix of both C and C++, with precious little documentation. As a result, there was an extremely steep learning curve for developers entering the project. This situation has changed slowly over the lifetime of the project, though there is still a daunting body of code for the new developer.

The initial plan for Mozilla was to release an OSS equivalent of the original Communicator product. This would be essentially the same product, with OSS code substituted for the proprietary modules. However, approximately a year after development begun, a decision was made by the MO to reset the Mozilla project around a new HTML layout engine, now called Gecko. This change was made to allow the browser to support the latest web standards and therefore remain competitive with competing browser technologies. It was also argued that the change to a cleaner code-base would make it easier for external developers to understand and contribute to the code-base. However, it essentially meant that a great deal of the work that had been performed on the old Mozilla code had to be abandoned. Although there is little doubt that the Gecko engine is far superior to the classic engine, the decision to reset the project caused considerable debate. The result of moving to Gecko was that a stable release of the code-base would take much longer. It would arguably have been better to produce a stable release of the project for production use, before moving to Gecko. However, this would have required maintaining an additional version of Mozilla in parallel, taking valuable effort away from the main project. It is likely that the move to Gecko is the correct solution in the long term, though this assumes that the Mozilla project will survive. The lack of a stable release is cited in [Ray99a] as the primary reason for the lack of success of the Mozilla project.

The main competition for the Mozilla project is the Microsoft Internet Explorer (IE) browser. Unlike Linux and Apache, which are competing with products that generate money for Microsoft, IE is supplied as a free product, although it is still closed-source. Consequently, one of the main advantages of the OSS approach, namely free redistribution, does not apply in this case. The fact that the source code is available is a great benefit to the OSS community and those concerned with non-restrictive licensing, but it is unlikely to mean much to end-users. Thus, in order to compete with IE on the desktop, Mozilla must achieve a significant advantage over IE, either through speed, features, and/or user-friendliness. Mozilla is currently aiming to become more standard-compliant than IE. This is almost certainly one of the reasons for the early adoption of the Gecko engine. However, the delay in the release of a stable version of the software has meant that IE is now by far the dominant browser on the Internet, a position that was once held by Netscape. Once again, Mozilla have arguably adopted the wrong strategy. The recent Galeon [Gal00] project demonstrates that it is possible to build a fast lightweight browser directly from the Gecko engine, without all the additional features such as Mail and News, etc. Thus, the MO could have released such a browser some time ago as a stable product, and added to this base over time. Other packages which supply News and Mail capabilities are already available in abundance from other sources. The result would have been a browser that was capable of competing with IE, a product that is often regarded as bloated and slow. It is likely that such a release would have significantly motivated the OSS community to get involved in the Mozilla project. However, by delaying the release of Mozilla until all of the features are available, Mozilla itself has become rather bloated.

According to [Hec99], the lack of enthusiasm for the Mozilla project in the OSS community is that it is perceived as a Netscape/AOL project which neither needs nor wants outside support. This has in part arisen from the apparent internal conflict of interest between the OSS Mozilla project and the branded Netscape/AOL product and the fact that Netscape/AOL are funding a large number of full-time Mozilla developers. However, this is a no-win situation; if Netscape/AOL were to put more funding and people into Mozilla, they will be accused of trying to dominate the project. However, if they were to withdraw support from Mozilla, they would be accused of exploiting the work of the volunteer developers for their own purposes, without giving anything back to the project. Furthermore, it is difficult for outside volunteers to compete in terms of time and resources with the full time funded developers. Although there is now significant commercial interest and involvement in Linux and Apache, the same problem has not arisen. This is largely because the development of Linux and Apache is not seen to be controlled by a commercial entity. Companies such as IBM and RedHat have gone to great lengths to ensure that their methods of working are consistent with the existing OSS communities to which they have become a part. The problem is further compounded by the poor perception of Netscape/AOL in the OSS community. The move to an OSS development model for Mozilla was only considered when the dominance of the Netscape browser over IE had collapsed. Therefore, there is some cynicism over the motives and commitment of Netscape/AOL to the OSS ideals, e.g. an underlying suspicion that Netscape may abandon Mozilla should the success of their commercial browser improve. Furthermore, the acquisition of Netscape by AOL did not sit well in the OSS community. AOL is despised almost to the same extent as Microsoft in the OSS community, due to the fact that they have vigorously defended their proprietary technologies, such as instant messaging, from reverse engineering by OSS or other means. As a result, the OSS community has tended to embrace other OSS browser efforts in preference to Mozilla.

The perception of the Mozilla project in the OSS community was dealt a further blow with the high profile resignation of Jamie Zawinski in 1999, one of the lead developers in the MO. In his resignation letter [Zaw99] he outlined his opinions on why the Mozilla project was failing to attract outside developer attention and failing to produce an end-user browser. Many of these have already been discussed above. However, the underlying reason for his resignation was the lack of innovation and creativity within Netscape and Mozilla. In his opinion, the Netscape/Mozilla browser had changed little since its initial creation around 1995. The original Netscape browser had changed the world by bringing networked communications to the masses. However, the resulting success of the company had led to stagnation; new releases of the Netscape browser became further apart, stability suffered, and the code became bloated. In his opinion, the failure of Mozilla to release even a beta product was symptomatic of the downhill slide within Netscape itself; the baggage that Netscape has brought along to Mozilla was responsible for dragging the project down.

6.2 Environmental Forces

Despite all the hazards and failures of the Mozilla project which we have highlighted, the project has achieved some notable successes. The Mozilla project has demonstrated that it is actually possible to take a closed-source product and turn it into an OSS product. This in itself motivated other companies, such as SGI and IBM, to adopt OSS development for a number of their technologies. It is almost certainly true to say that Mozilla was in part responsible for the current rise in profile and popularity of OSS development. Although the project has not attracted the level of outside participation that was anticipated, it is certainly true that there has been a lot of outside feedback into the project. It is probably fair to say that this feedback has led the project in the right direction. The Mozilla project has also created a number of useful ancillary tools, such as the OSS Bugzilla database, and the Bonsai interface to CVS. Bugzilla is now used in many other OSS projects, including the RedHat Linux distribution. Finally, the long-term prognosis for Mozilla appears to be improving. The Milestone releases have been gradually improving in terms of feature completeness and stability. This has been helped by motivational events such as the Bug Day and Bug A-Thon. As of late 2000, the code has reached beta quality, with a planned release in 2001. The derived Netscape 6 browser is also reaching final release status.

6.3 A Mozilla Ecosystem

Our model of the Mozilla environment is illustrated in Figure 3. We have isolated two different kinds of agents, namely *module owners* and *peers*. We have also introduced the concept of a *group* of agents: a subset of the module owners are part of the Mozilla Organisation (MO) group. These agents may poses additional attributes or responsibilities over the agents which are not part of this group. In the Mozilla project, the module owners who are members of the MO are the employees of Netscape/AOL. It is worth noting that we are only modelling the development process. Thus, although the actual MO consists of many individuals and provides many different services, we are only concerned with the effect of the MO on code development in our model.

Agents possess quantities of resource (i.e. code) and exchange this resource by message passing. Agents are individually responsible for accepting or rejecting messages, i.e. there are no ‘voting’ messages as in the Apache model. This model is similar to that of Linux, though the decision making process is decentralised. Decisions in the ecosystem are made by module owners who are members of the MO. Certain MO agents have extra responsibilities such as an *arbiter* who decides between competing solutions, and a *release manager* who gathers together code, e.g. for the Milestone releases. There are many more roles than these, such as quality assurance (QA), port management, etc. These roles are permanently attached to a particular agent until the agent leaves the ecosystem.

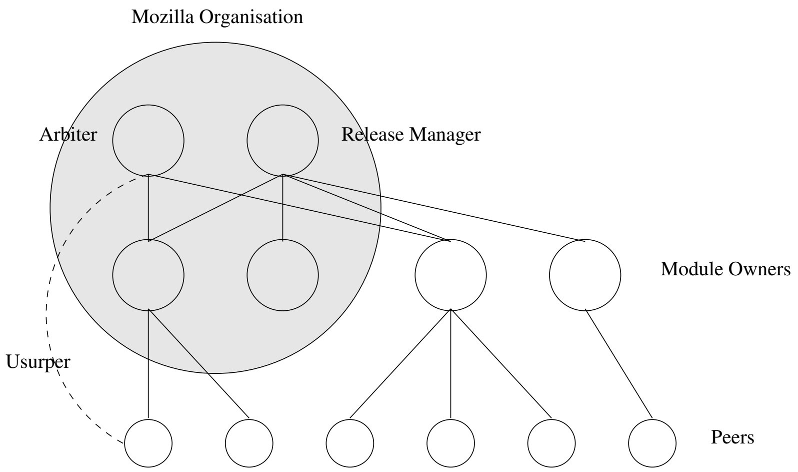

Figure 3: The Mozilla Ecosystem

The module owner agents have a direct responsibility for a quantity of resource. This resource will grow over time as messages are accepted from the peer agents. The responsibility of a module owner is to have this code accepted by one or more MO agents. Module owners will occasionally join and leave the MO group, corresponding to developers which are employed by Netscape/AOL. As discussed earlier, it is occasionally possible for a peer to usurp a module owner. This occurs in our model when a message from a peer bypasses the module owner and is accepted by a MO agent. At this point, the peer will have the option of promotion to a module owner. The original module owner will either continue as a competitor, become a peer to the new module owner, or leave the ecosystem. Thus, unlike our earlier models, agents can be demoted as well as promoted in the ecosystem.

New agents will be created as peers to particular module owners. Peers possess smaller quantities of resource than module owners, and will frequently leave the ecosystem. Over time, as the code-base grows, peers will be promoted to module owners to maintain quantities of resource. This is separate from usurping a particular module owner. The promotion of peers is controlled by an MO agent, and is based on the level of activity of a particular peer over time (i.e. reputation).

It is clear that our ecological model of Mozilla has many features in common with both the Linux and Apache models presented earlier. The novelty in this model is the Mozilla Organisation group, essentially representing commercial influence on the development process. It will be interesting to study the extent to which this influence dominates the development process over the lifecycle of the ecology. By experimenting with the role of the MO in the ecology, we hope to be able to demonstrate quantitatively how the current Mozilla development process can be improved.

7 Conclusions

This paper is intended to serve as the first step in our ongoing project to model the behaviour of a variety of information ecosystems. Although the focus of this paper is on the OSS community, it is our intention that the techniques that we develop can also be applied to a wide variety of other information ecosystems. The experiments that we intend to perform on our OSS models are outlined below.

1. A study of the operation of the ecosystem under normal conditions. This is a straightforward simulation of the behaviour of the ecosystem over time. The most important consideration for this experiment is the instantiation of the ecosystem, given the large number of parameters involved, e.g. agent birth, activity, and mortality rates. Our intention is to initiate each ecosystem at a relatively small size, and tune the simulation parameters to match the actual observed rates of growth, based on studies such as [MFH00]. Once we have obtained a result which approximates the existing behaviour of the ecosystem, we can run the simulation into the future and examine the growth potential of each ecosystem. Our simulations will allow a direct comparison of the three styles of development, and should highlight potential bottlenecks in each. We have previously identified a number of techniques for eliminating bottlenecks in each ecosystem, e.g. increasing the depth of the developer hierarchy, relaxing voting conditions, relaxing grouping constraints. Thus, using our simulation, we can directly study the effects of these techniques. The result of these experiments should be a list of suggestions for each ecosystem in order to promote future growth.

2. A study of the response of the ecosystem to threats and hazards. In this experiment, we introduce artificial hazards into each ecosystem and examine their effects. There are many different ways in which this can be done, though we intend to concentrate on a small number corresponding to real scenarios which we have outlined earlier. The first scenario is the sudden removal of agents central to the each ecosystem. This is intended to address concerns such as the removal of Linus from the Linux, or the removal of Netscape/AOL (i.e. the MO) from Mozilla. Our conjecture is that ecosystems with a loose structure (Linux) will adapt much better than those with a rigid structure (Mozilla). However, the extent to which the ecosystem can recover can only be observed through simulation. The second scenario is the sudden removal of resource from the ecosystem. This is intended to correspond to such events as code forks, or patent issues. Our conjecture is that larger ecosystems will recover more quickly than smaller ecosystems, though the actual response rates for the different styles of organisation will be of primary importance. The final scenario is the introduction of hostile agents into the ecosystem, whose purpose is to disrupt development, through techniques such as resource hogging, message flooding, etc. This is intended to represent threats such as unwanted commercial influences and security aspects such as viruses. Once again, the response of the ecosystem will be of primary importance. The result of these simulations will be a characterisation of potential weaknesses in each ecosystem, and a list of improvements to each which would enhance the robustness and response to such threats.

3. A study of different lifecycles in the ecosystem. The intention of our experiments is to understand the interactions between different lifecycles in our ecosystems. However, we have primarily concentrated on the lifecycles of the (human) developer agents, while lifecycles in the underlying code (resources) have been largely concealed. In order to expose these lifecycles, we need to take a different view of the system. An appropriate view can be obtained by reversing the roles of the developers and code,

with the result that developers are modelled as resource and the code is modelled by the agents. In this experiment, we will construct models of each system using this alternative view and observe the resulting behaviours. We expect that the result will be a smaller and less dynamic ecosystem than the previous view; where birth and death of agents will tend to revolve in cycles relating to different code releases, and only a minimal exchange of resource (i.e. developers) between agents. Nonetheless, this view will expose problems such as long release cycles, bloat, and stale code. We hope that this view will enable further insight into the dynamics of each ecosystem, though the effect of this view on our understanding remains to be seen. We note that in all information ecosystems, there will generally be many different choices of resource, each potentially highlighting a different aspect of the system.

4. A study of diversity arising in each ecosystem. In this experiment we will identify and explore niches which arise naturally during ecosystem lifecycles. This requires a more subtle finer-grained set of observations than in previous experiments, based on observing behaviours of individual agents rather than agent populations. The more obvious niches that may arise are those such as groupings of closely connected developers, or parts of the hierarchy which exhibit starvation. However, there are also less obvious niches which can emerge spontaneously from the interactions of agents over time. Such niches tend to arise in all systems of decentralised interaction, e.g. business models [AEY00], and tend to perpetuate for long periods of time, even though they may have arisen from purely random events. We hope that the outcome of these experiments will be a more detailed understanding of the underlying dynamic of each ecosystem, highlighting more subtle properties such as missed opportunities and potential disputes or splits.

We are currently building a number of tools which will enable simulations and empirical measurements of our ecosystems to be performed [Vas01]. We have defined a language based on temporal logics, in the style of MetateM [BFG+95], for describing agent behaviours. This language also contains message-passing primitives which facilitate the construction of large multi-agent systems. Our specifications are executable through an interpreter, providing us with a method for performing controlled simulations. The temporal logic underpinning the language will allow verification techniques, such as model checking [CGP99], to be performed on sections of the ecosystems. This will enable us to verify properties over all possible behaviours, and therefore ensure that the results of our simulations do not simply correspond to special-cases. This framework should be powerful enough to represent all of the ecosystems in this paper.

References

[AEY00] Robert Axtell, Joshua M. Epstein, and H. Peyton Young. The Emergence of Classes in a Multi-Agent Bargaining Model. Technical report, Center on Social and Economic Dynamics (CSED), The Brookings Institution, February 2000. Available at: `http://www.brook.edu/es/dynamics/papers/classes/`.

[BFG+95] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. MetateM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[ČB99] Davor Čubranić and Kellogg S. Booth. Coordinating Open-Source Software Development. In *Proceedings of the 8th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 61–65, Stanford, California, June 1999. IEEE Computer Society.

[CGP99] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[DO99] Chris DiBona and Sam Ockman, editors. *Open Sources: Voices from the Open Source Revolution*. O'Reilly, February 1999.

[Fie99] Roy T. Fielding. Shared Leadership in the Apache Project. *Communications of the ACM*, 42(4):42–43, 1999.

[Gal00] Galeon Development Team. Galeon Manifesto. Available at: `http://galeon.sourceforge.net/info.html`, 2000.

[Gla00] Robert L. Glass. The Sociology of Open Source: Of Cults and Cultures. *IEEE Software*, pages 104–105, May 2000.

[Gro99] Apache Group. About the Apache HTTP Server Project. Available at: `http://httpd.apache.org/ABOUT_APACHE.html`, February 1999.

[GT00] Michael W. Godfrey and Qiang Tu. Evolution in Open Source Software: A Case Study. In *Proceedings of the International Conference on Software Maintenance (ICSM2000)*, pages 131–142, San Jose, California, October 2000. IEEE Computer Society.

[Hec99] Frank Hecker. Mozilla at One: A Look Back and Ahead. Available at: `http://www.mozilla.org/mozilla-at-one.html`, April 1999.

[Keg99] Dan Kegel. On Mindcraft's April 1999 Benchmark. Available at: `http://www.kegel.com/mindcraft_redux.html`, 1999.

[Lin99] Owen W. Linzmayer. *Apple Confidential : The Real Story of Apple Computer, Inc.* No Starch Press, March 1999.

[McK99] Niall McKay. Apache-IBM Marriage Bears Children. Available at: `http://www.linuxworld.com/linuxworld/expo/lw-apache.html`, February 1999.

[MFH00] Audris Mockus, Roy T. Fielding, and James Herbsleb. A Case Study of Open Source Software Development: The Apache Server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE2000)*, pages 263–272, Limerick, Ireland, June 2000. ACM Press.

[Net00] Netcraft. The Netcraft Web Server Survey. Available at: `http://www.netcraft.com/survey/`, October 2000.

[Per97] Bruce Perens. The Open Source Definition. Available at: `http://www.opensource.org/osd.html`, June 1997.

[Ray98] Eric S. Raymond. Homesteading the Noosphere. Available at: `http://www.tuxedo.org/~esr/writings/homesteading/`, April 1998.

[Ray99a] Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly, October 1999. Also available at: `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/`.

[Ray99b] Eric S. Raymond. The Magic Cauldron. Available at: `http://www.tuxedo.org/~esr/writings/magic-cauldron/`, May 1999.

[Sta98] Richard M. Stallman. Why "Free Software" is better than "Open Source". Available at: `http://www.gnu.org/philosophy/free-software-for-freedom.html`, 1998.

[Vas01] Wamberto Vasconcelos. Verification of executable temporal specifications for large multi-agent systems. In preparation, Jan 2001.

[Zaw99] Jamie Zawinski. Resignation and Postmortem. Available at: `http://www.jwz.org/gruntle/nomo.html`, March 1999.