

Synchronize and Stabilize vs. Open-Source

An analysis of 2 team-organization models when developing consumer-level shrink-wrapped software.

Computer Science 95.314A Research Report

By: *Shahzad Malik (219762) and Jose Ruben Palencia (277158)*

For: *Francis Bordeleau*

Carleton University

Ottawa, Ontario, Canada

December 6, 1999

1 Introduction

Effective team organization is a vital part of any successful software product. Having a team of the world's best programmers or software engineers is extremely important, but if this group isn't properly organized the project is destined for failure from day one. With the extreme (financial) success experienced by software giant Microsoft Corporation, it is fitting to look at the Synchronize and Stabilize team model used on almost all of their consumer level products such as Microsoft Office, Windows 95/98, and Windows NT. The second team model this paper will look at is the Open Source model, which has recently gained recognition by both the rise of the Internet and the popularity of the Linux open-source operating system. While each model can apply to both a life-cycle or team organization, this paper will focus on the team aspect which deals with issues such as team cooperation, productivity, communication, and team member responsibilities. We will provide an overview of each team model, do a comparison of Microsoft's Synchronize and Stabilize approach with the Mozilla Organization's Open-Source approach, and then offer some insights on what makes each model work, what the pitfalls are, and how adaptable each model is to any given project.

2 Objectives

- Provide an overview of the Synchronize and Stabilize and Open-Source teams, and how they compare to traditional team organization models.
- Compare Microsoft's Synchronize and Stabilize model to Mozilla Organization's Open-Source model, which effectively allows us to compare the development of Microsoft's Internet Explorer and Netscape's Navigator web browser.
- Provide a summary analysis of the 2 models, based on what key components make each model work, what the pitfalls of each model are, and whether each model can be successfully applied to any given organization or software project.

3 The Synchronize and Stabilize Team

The idea behind the Synchronize and Stabilize model is simple: continually synchronize what people are doing as individuals and as members of parallel teams, and periodically stabilize the product in increments as a project proceeds, rather than once at the end of a project [3]. Microsoft has been using this approach to software development and team organization since the late 1980s on many of its popular products, including Office, Publisher, Windows 95/98, and Windows NT, and has achieved great success as a result.

The typical team layout can be seen in Figure 1. Since Microsoft's products are typically consumer driven (rather than specific to one particular client), each product is initiated by a small, focused group of Product Managers (marketing specialists) which develop a "vision statement" for the product [3]. This vision statement, developed using extensive customer feedback, clearly outlines the goals and features that the product must support, and prioritizes these features based on how critical they are to the finished software. The vision statement is then passed along to a group of Program Managers, which use this information to define a functional specification outlining feature functionality, architectural issues, and component

interdependencies. After the functional specification is defined, the Program Managers coordinate schedules and arrange the Feature Teams (Figure 1) such that each contains approximately 1 program manager, 3-8 programmers, and 3-8 testers which work in parallel 1:1 with the programmers.

The interesting output from this planning phase is the initial functional specification. The Program Managers specifically avoid trying to cover all the minute details of the desired features, or lock the project into any fixed set of features. Instead, based on past experiences where feature sets have varied by 30% or more, the functional specification remains relatively open-ended such that the Feature Teams themselves can evolve the product during development [3]. This is a key component of the Synchronize and Stabilize approach to team organization. In Figure 1, each parallel Feature Team operates in a process similar to a traditional Democratic Team [1], such that creativity and autonomy of design is fostered, without too much bureaucracy from above. Paths of communication flow freely and democratically between the 3-8 developers and testers, yet the Synchronize and Stabilize team still maintains a sense of order and direction by the presence of the Program Managers, all of whom are experienced in both technical and managerial issues.

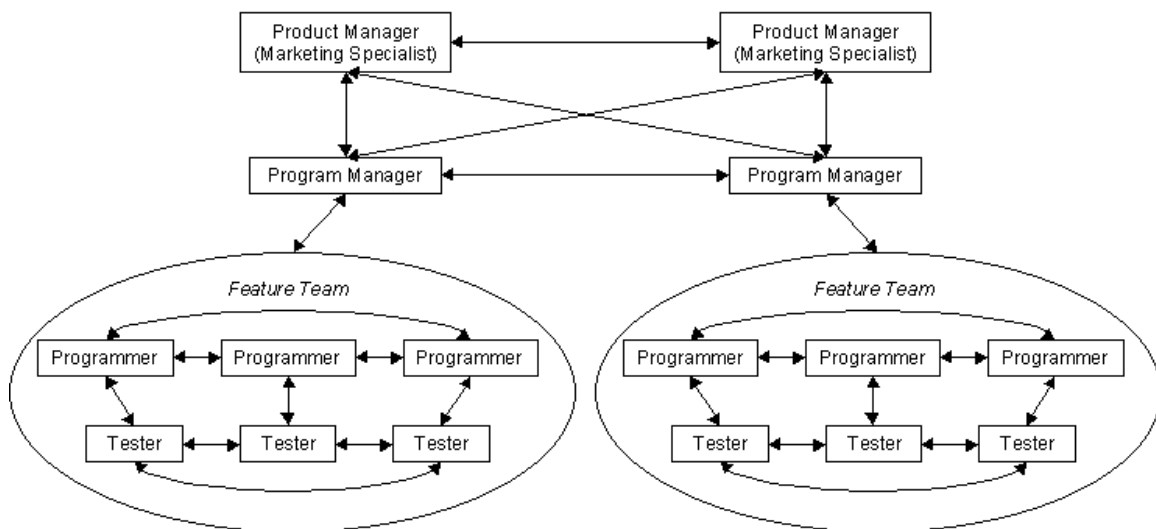


Figure 1: Overview of the Synchronize and Stabilize team-organization and decision-making approach

After the planning phase is completed (which typically takes between 3 and 12 months depending on the size and complexity of the product), the project shifts into the development phase. At this point, projects are divided into 3 or 4 subprojects and milestone “releases”, where each team goes through a complete cycle of development (feature integration, testing, debugging). During each of these cycles, programmers are free to develop their components and features autonomously, provided they could be integrated flawlessly at the end of each day with all the other components. The 1:1 programmer/tester ratio allows this daily synchronization step to occur relatively flawlessly throughout most of the development cycle. However, sometimes build errors do occur, and each team imposes their own penalties for “breaking the build” – ranging from a \$5 fine, to wearing goat horns for the day, to taking over responsibility for creating the daily builds until someone else breaks it [2].

At the end of each subproject, the evolving product is thoroughly tested and fixed (stabilized) such that all the required features for the milestone are functioning flawlessly, and the team can then proceed to the next milestone release (or ship date) [3].

The Synchronize and Stabilize team process can be summarized as follows:

- Feature oriented
 - Synchronize (daily build) and Stabilize (fix errors at end of each milestone, such that the required set of features is completely functional)
 - Product Managers develop a vision statement and feature list based on customer input
 - Program Managers develop an initial functional specification based on the vision statement
 - Program Managers create schedules and parallel feature teams of 3-8 developers and testers based on the functional specification
 - Team members can work autonomously, thus maintaining some creative freedom on the project, provided their work can be combined successfully into the daily builds
 - Teams develop their own playful penalties for breaking the daily build, which forces a certain amount of discipline amongst a team, while at the same time remaining democratic
- Planning phase (3-12 months, depending on size of project)
 - vision statement
 - specification
 - schedule and feature team formation
 - Development phase (6-12 months)
 - subproject I: most crucial 1/3 features, milestone release I
 - subproject II: second 1/3 features, milestone release II
 - subproject III: final (least critical) 1/3 features, milestone release III --- code complete
 - Stabilization phase (3-8 months)
 - internal testing within the company
 - external testing at "beta" sites
 - release for manufacturing, which involves the final release of a "golden master" disk and documentation

4 The Open-Source Team

While its been around for almost 20 years in some form, the concept of Open-Source software development has only recently gained widespread recognition as a result of the recent rise of the Internet as a communication and information medium. The proper definition of Open-Source is continuously debated, but one aspect that is agreed upon is the fact that the source code for a product developed using open-source licensing is fully accessible to any end-user [8].

In terms of team organization, however, there is no clearly defined methodology as to how a piece of software should be developed under the open-source approach. In the case of the popular open-source Linux operating system, development progressed almost by purely unexpected excitement and synergy between developers after one programmer released the source code to a Unix-like operating system that he was developing as a hobby. In other cases, such as Netscape's web browser, the core software was originally developed as a proprietary commercial product for over 5 years until the company decided to release it to the general public under open-source terms. In all cases, the most fascinating aspect of teams developing the software is the fact that team members could potentially be working on components in completely different parts of the world. The need for proper organization thus becomes a major concern.

Figure 2 shows 2 different models that are commonly used by teams developing open-source software. In all cases, the key point to remember is the fact that each of the Programmers can be (and usually are) scattered across the world, and the set of all Programmers is a subset of the product's User community.

- Figure 2a is what we call the Democratic Open-Source team, and is the methodology which is used on many open-source projects that are just getting started. This model is almost identical to a traditional Democratic Team approach [1]. However, a key difference is the fact that a traditional Democratic Team stresses communication and coordination between team members, whereas the Democratic Open-Source Team leaves coordination and synchronization as a purely optional facet of development.
- Figure 2b is what we call the Collective Open-Source Team. This model shares a lot in common with a traditional Chief Programmer team or even a Synchronize and Stabilize team, but with the added complexity that the Collective (or Project Owner [12]) can be any number of developers, not necessarily one Chief Programmer or Program Manager. This approach is currently being used by the Linux kernel developers, as well as the Mozilla Organization which now oversees the development of Netscape's web browser.

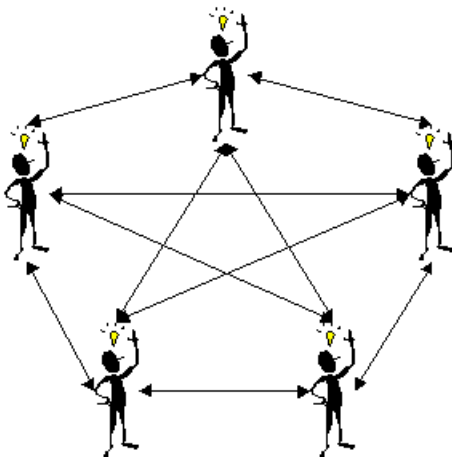


Figure 2a: Overview of a Democratic Open-Source team-organization and decision-making approach (5 developers shown). Note that each line of communication is optional, and not necessarily enforced.

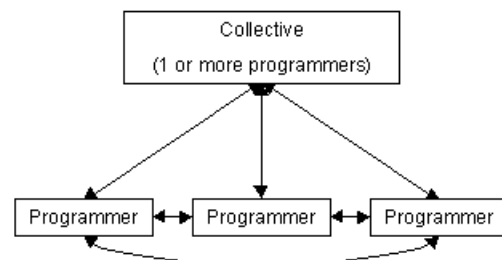


Figure 2b: Overview of a Collective Open-Source team-organization and decision-making approach (3 developers shown). Note that the lines of communication are always 2-way. Also, each Programmer can itself have a hierarchy of other developers.

One thing worth noting in the Figure 2 diagrams is the notion of the Programmer; the Programmer need not be one individual. In the case of RedHat's distribution of Linux, the company can be considered the Programmer. Internally they are free to use their own team-organization and life-cycle models, but by the standards of open-source development they are still required to release their code changes back into the hands of other developers. On larger projects, each Programmer or set of Programmers may be separated based on subsystems, which enforces a much more modular design and code base as well as smaller, more focused sub-teams.

Unlike the Synchronize and Stabilize model, the majority of open-source projects have sketchy marketing requirements, no system-level design, little detailed design, virtually no design documentation, and no system-level testing [13]. One or two programmers initiate most open-source projects, with functional necessity, rather than financial, driving the development of the product. As a result, most open-source projects start off in a hobbyist code-and-fix approach, with no formal requirements or specifications being defined. Eric Raymond captures the beginnings of an open-source project perfectly when he says:

“When you start community building, what you need to be able to present is a *plausible promise*. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future.” [11]

Once an open-source project is initiated, there are 2 popular ways for it to proceed. In the case of many early Unix tools or previously proprietary products such as Netscape's web browser, the majority of design and coding was done in isolation by a team of programmers until the software progressed to the point where it was nearly complete and ready to use. At this point, the software and all of its associated source code would be released to the public, where any user could make their own code and feature-set improvements. Other projects, such as the Linux kernel, follow more of a rapid-prototype approach, where a single programmer releases an initial piece of software with minimal functionality, and the rest of the product is developed in the hands of users. The initial programmer then assumes the role of the Collective or Project Owner through which all outside code submissions and updates must pass, and through which new iterations and releases to the public are made. Clearly, the development process on open-source projects is reminiscent of Brooks famous suggestion: *grow, don't build, software* [15].

A typical iteration of an open-source project proceeds as follows:

- A Programmer performs design and coding (on user requested tasks), in either total autonomy or in conjunction with other developers.
- Debugging and testing are performed in parallel with design and coding, by either other programmers or general users
- Bug fixes and new bug findings, as well as feature requests, are passed along to the Collective or Project Owner
- A new release is then put together, where code submissions from other developers are integrated into the code base, new found bugs are tracked, and feature requests by users are reviewed
- As soon as the new release is made, the implementable feature requests and bug findings are transferred to an open task list, and developers may volunteer to implement desired tasks for the next iteration

With open-source teams potentially spread across the world, communication among team members is critical. Clearly, the Internet is a key component to the success of open-source projects. Developer mailing lists, email, and newsgroups are currently the principle means of communication among project contributors, while web and FTP sites provide a quick and easy way for users and developers to get the latest code releases and bug fixes [12].

Since an open-source project typically allows anyone to be a part of the development of the product, the quality and experience of team members and their code typically varies greatly. As a result, maintaining a consistent code base becomes increasingly difficult. However, in teams which employ the notion of a Collective or Project Owner that controls what changes are included in a project release, the issue of code quality and consistency becomes less of an issue.

The Open-Source Team can be summarized as follows:

- User driven, where users chip in their own improvements
- Decentralized teams are distributed across the world, using the Internet as the primary means of communication and coordination
- Initial development typically follows a Code-and-Fix approach
- As feature requests are made and the product evolves, development follows an Iterative and Rapid Prototype approach
- Testing and debugging is performed by the user and developer community in parallel to development, and bug findings are submitted to a central authority where they are tracked
- New feature requests and bug fixes are implemented on a volunteer basis

5 Comparison: Microsoft vs. Mozilla

Most software related comparisons usually come down to Microsoft versus Everyone Else, and team organization is no exception. The Synchronize and Stabilize model has been developed and refined at Microsoft Corporation, with no well-known cases of its use elsewhere. In this

section, Microsoft's Synchronize and Stabilize team model, used for the development of Internet Explorer, will be compared with the open-source model used by the Mozilla Organization at Netscape. Mozilla is a group within Netscape that is chartered to act as the Collective (see Figure 2b) for the publicly available Netscape Navigator web browser source code [10].

While there is no clear-cut method to assess the quality of any given software development team, a few key components can be measured in order to determine a team's strengths and weaknesses. Three important things to look at include the quality and experience of the people involved, distribution of responsibilities, paths of communication and coordination between team members, and scalability of the team organization as the project grows. Other important but less predictable things to consider are psychological issues, such as team morale and individual productivity.

People

Microsoft (Synchronize and Stabilize)

Using financial resources as a major incentive, Microsoft's successful use of the Synchronize and Stabilize model is largely a consequence of the quality of people the company hires. Product Managers are business and marketing specialists with a solid grasp of technology, such that they can spot and grasp opportunities when they appear. Similarly, Program Managers are extremely talented individuals both technically and managerially. They must be able to take high-level product requirements and descriptions from the Product Managers, and effectively develop a detailed schedule of technical tasks for the programming teams.

Mozilla (Open Source)

As with all open-source projects, the developers contributing code to further develop Netscape's software are a subset of the people using the product. These self-selected developers are thus motivated not by money, but by both an interest and ability to contribute to a product they use on a regular basis. This remains generally true of open-source developers even when they are being paid a salary to "hack" open source projects [11].

How they compare

Clearly, the motivations for development between Microsoft's developers and Mozilla's developers are completely opposite. Finances drive Microsoft, and the company uses its financial muscle to hire the most talented people possible. Open-source projects, on the other hand, consist of developers that contribute purely for the sake of interest and enjoyment in developing software that they will use on a regular basis. Similarly, the quality of developers between the two processes can vary greatly. Microsoft relies on a thorough screening and interview process in order to hire only the best people, whereas open source groups such as Mozilla allow anyone interested in the project to contribute.

Responsibilities

Microsoft (Synchronize and Stabilize)

After developing the initial vision statement, requirements specification, and development schedule for a product, the Product and Program Managers take on a typical managerial role for the remainder of the project. Aside from usual paperwork duties, they rigorously track the progress of the daily builds by monitoring how many bugs are newly opened, resolved, fixed, and active. To create these daily builds, a designated developer, called the project build master, is responsible for generating a complete build of the product using the master source code. Each programmer is responsible for his or her assigned tasks (as well as evolving new features or technologies), and is required to check-in their changes to the master source code on a regular basis. Testers work closely with their one assigned programmer, to verify that the changes the programmer has made will be integrated flawlessly into the master build.

Mozilla (Open Source)

This Mozilla group acts as the Collective for the Netscape browser source code, and as a result collects changes, helps authors synchronize their work, and periodically makes new source

releases which incorporate the best work of all contributors as a whole [10]. Mozilla subdivides responsibilities by *module owner*, which is an individual that assumes responsibility for a large component or subsystem of code. Code contributors thus choose a module that interests them, and then make changes, submissions, bug reports, etc. to the associated discussion forums or module owner.

How they compare

With respect to responsibilities, the Synchronize and Stabilize and Open Source methodologies are remarkably similar. Both rely on the notion of a central authority through which all changes must pass through to be included in any release.

Communication and Coordination

Microsoft (Synchronize and Stabilize)

Even though Microsoft creates large teams consisting of a set of smaller Feature Teams, the multiplicity of communication channels can still become relatively high. For example, in a Feature Team with 8 programmers and 8 testers, there is a 1:1 communication path between each programmer and tester, 36 paths of communication between the programmers, and 36 paths of communication between the testers. However, with Microsoft's selective hiring processes, the company can create "near-egoless" teams of individuals which will work well together.

Additionally, with the presence of a Program Manager (see Figure 1), each Feature Team maintains a sense of order when disagreements do occur. In this way, the Synchronize and Stabilize Team encompasses the best of both worlds: the independence of a Democratic Team [1] combined with the order of a Chief Programmer Team [1]. Finally, Microsoft teams work at a single physical site with common development languages, common coding styles, and standardized development tools, which helps teams communicate, debate design ideas, and resolve problems face to face [3].

Mozilla (Open Source)

Brooks' Law states that the complexity and communication costs of a project rise with the square of the number of developers, while work rises linearly [16]. However, a popular assertion is that open-source projects defy this statement [12]. Mozilla's concept of a *module owner* naturally leads to a modular design, which directly minimizes the *required* paths of communication between developers. With open-source projects it's not necessary, and regularly discouraged, for contributors to understand the details of each and every subsystem. Instead, developers are expected to concentrate on a particular area of interest, while maintaining a high-level view of the entire project [12]. The Mozilla group thus operates a central collection of newsgroups and mailings lists on their web site, separated by modules, through which effective communication between developers may occur. Mozilla also coordinates bug lists and attempts to provide "roadmaps" to the code, and to projects based on the code. Mozilla also helps developers reach consensus, and thereby provides direction and coordination for future improvements [12].

How they compare

When it comes to communication and coordination, it's difficult to assess whether the two organization methods are nearly identical or completely different. On the one hand, both Synchronize and Stabilize and Open Source stress small democratic teams, each working in a modular fashion, with creativity on certain portions of the project's design. On the other hand, Synchronize and Stabilize stresses face to face discussion with other team members, as well as a 1:1 programmer/tester ratio, while Open Source teams rely on Internet newsgroups and email for communication. Ideally, face to face communication would be preferable, since its immediate as opposed to the delays associated with communicating through the Internet.

Scalability

Microsoft (Synchronize and Stabilize)

Microsoft relies heavily on the planning phase of development to schedule, prioritize, and allocate resources to project tasks. With close to 20 years of experience in the PC software industry, the company is able to accurately predict their development cycles. As a result, the Synchronize and

Stabilize teams aren't necessarily equipped to handle large-scale changes in the team structure after the project has already been initiated, both for synchronization and budgetary reasons. However, the Synchronize and Stabilize process, in general, can be scaled to larger projects quite nicely at the preplanning phase, since more and more Feature Teams of 3-8 developers and testers, with associated Program Managers, may be added as required without adding much more to the complexity of the project.

Mozilla (Open Source)

The general openness of the Mozilla Organization to contributors shows the scalability and flexibility of open source projects. New contributors may come and go as a project progresses, since there is no fixed contributor associated with any one piece of code. If one contributor cannot complete an important task or bug fix, another one will almost always be nearby to finish the job. The one major exception to this are the module owners. However, due to the dynamic nature of open-source projects, and the large pool of talent available, module owners can easily hand off their position to other worthy module contributors if they can no longer fill the position. Mozilla Organization likes to refer to their people structure as a Benevolent Dictatorship, whereby an "elected" or worthy module owner will represent the changes and submissions of a module to Mozilla, until that module owner either loses interest or is voted out of his/her position by other module contributors.

How they compare

Clearly the success of the Synchronize and Stabilize model relies heavily on the ability of Program and Project Managers to develop a functional specification, schedule, and budget in the initial phases of development. This leads to a relatively fixed team structure as the project progresses through development. The Open Source model, on the other hand, remains open at all times, allowing contributors to come and go as they please based on personal interests in the project.

6 Analysis of the Synchronize and Stabilize Team

People

Advantages

- With Microsoft's financial resources, the company can pick and choose the smartest, most qualified developers
- Product Managers know the technology business extremely well and can spot new opportunities as they appear

Disadvantages

- Finding good Product Managers and Program Managers is a difficult task, since they require multi-talented individuals that know about technology, business, and managerial issues

Responsibilities

Advantages

- Clear separation of duties and detailed scheduling leads to a focused development process and clearly defined milestone release dates
- Daily builds guarantee the existence of an always working, usable product
- 1:1 programmer/tester ratio reduces bugs which can easily be overlooked by a developer
- Allowing developers some creative freedom to evolve new features or technologies keeps developers interested and dedicated to the project

Disadvantages

- Daily builds create a high-pressure environment for developers and testers
- Allowing tasks to be developed autonomously can lead to timely-feature integration difficulties if a developer leaves midway through a project

Communication and Coordination

Advantages

- Teams can be specially selected such that members work well together, which leads to a higher level of productivity
- Presence of Program Managers with a good grasp of technical issues maintains a sense of order, but without inhibiting creativity among developers
- Centralized development leads to better communication and consistency between developers

Disadvantages

- Despite careful selection of developers, teams may not always work well together, which clashes with Microsoft's attempt to create democratic Feature Teams

Scalability

Advantages

- Scalability to larger projects can occur smoothly at the planning phase, provided the project has experienced Program Managers

Disadvantages

- Scalability on an already initiated project is difficult with regards to budgets and communication, since teams need to be carefully selected to ensure teams which work well together

Other issues

Advantages

- Daily builds ensure all components work together
- Evolving the product allows teams to get an early insight into the operation of the product
- Suits the constantly changing PC software industry, since having a shippable (runnable) product at all times prevents competitors from stealing market-share
- Promotes individual creativity
- Daily synchronization promotes communication and coordination

Disadvantages

- High-pressure atmosphere is created by the daily build step
- Requires an extremely thorough scheduling stage
- Focus on creating sellable products doesn't necessarily equate to quality products

7 Analysis of the Open Source Team

People

Advantages

- Contributors are usually volunteers, which means focus is on the quality and usefulness of the product, rather than making money
- Anyone can contribute to the development, resulting in a huge base of potential developers

Disadvantages

- Potentially inconsistent experience levels among developers, leading to inconsistent code quality
- Distributed nature of development (as opposed to the centralized nature of Microsoft's approach) leads to difficulties in communication between developers

Responsibilities

Advantages

- Presence of a centralized integrator (eg. Mozilla) allows proper synchronization of changes
- Contributors can work on modules or subsystems which they are interested in
- Potential for hundreds of developers means almost any programming problem or bug can be overcome; *"given enough eyeballs, all bugs are shallow"* [11]

Disadvantages

- Allowing developers to choose modules based on interest (rather than experience) can lead to code submissions which are experimental rather than solidly implemented
- Lack of formal schedules or prioritization of tasks leads to unpredictable release dates

Communication and Coordination

Advantages

- Separation of code into modules leads to a modular design, and modular teams of developers
- Use of a central authority for design and coordination allows conflicts to be effectively resolved

Disadvantages

- Effectiveness of Internet based communication over face to face communication is still relatively untested
- Potential for hundreds of developers may lead to overlapping of tasks and inefficient use of human resources (ie. 2 people could potentially be working on the same feature or bug fix)

Scalability

Advantages

- The project can grow as needed with regards to new people, since tasks are separated more by interest and self-selection, rather than a schedule and budget

Disadvantages

- Since developers typically contribute to open-source projects based on interest, projects which aren't cutting-edge or exciting may have a hard time drawing contributors
- As projects become more complex, more formal commercial structures will evolve [12]

Other issues

Advantages

- Driven by user needs, rather than finances, which leads to a more refined product
- Most open-source development is done free of charge by dedicated users
- The developer community creates an "Intellectual Olympics", where some of the world's top engineering minds compete -- not for venture capital, but for impressing their peers [9].
- Successful products such as Linux, Netscape Navigator, Apache Web Server, Star Office, and Sendmail prove the viability of the open-source team process

Disadvantages

- Model doesn't stress clearly defined milestones for each release, since functionality is added on a volunteer basis
- Code quality and consistency is an issue, based on inconsistency in developer's experiences
- Success in feature-oriented, non-system-level software is still questionable, such as educational software, entertainment software, etc.
- No clear-cut "end" of project is defined; projects progress until the software is stable and feature complete, which is usually signified by a fall in the number of contributors

8 Conclusion

The ultimate question to ask regarding the above team organization models is what results in a better piece of software, released on time and under budget, fulfilling the end-users requirements? Additionally, what leads to a more productive team, with high morale among its members?

The fact that Open Source teams are driven by a desire to create useful software, rather than finances, is sure to make traditional managers nervous. Clearly, the lack of concise scheduling or budgetary constraints doesn't allow an Open Source team to be a viable option for just any piece of software. The recent successes in Open Source teams have come from system-level

software, such as operating systems (Linux) and servers (Apache). Products such as office suites (Star Office) or web browsers (Netscape) have only recently been released under open source terms, after largely being developed as proprietary packages for the past few years. Only time will tell if these products can be successful Open Source products, but the adaptability of Open Source to just any piece of software doesn't look promising. Open Source products such as Linux, Apache Web Server, or Netscape Navigator are software with large user bases. These products demand stability and constant updates as platforms and hardware evolve, making them ideal open source projects. Feature-rich software on the other hand, such as video games or educational tools, are placed into "retirement" normally within a year of their release. Combined with a much smaller user base, these products cannot possibly warrant a continuous development cycle. In these cases, the Synchronize and Stabilize model wins hands down, since it employs a much more strict development schedule and budgetary constraints.

When it comes to satisfying user requirements however, Open Source teams clearly have the upper hand. By the simple fact that open source developers are a subset of a product's user community ensures that user requirements are being met. Microsoft's Synchronize and Stabilize model relies more on what the Project Manager has determined as being critical requirements, based on past customer feedback.

With respect to team productivity and morale, no one model is a clear winner. The Open Source model definitely has a lot going for it, since team members are completely self-selected, working on portions of a piece of software which they personally want to see succeed. Rather than seeing development as a nine-to-five affair, open-source programmers tend to develop their portions of a project in their spare time, at their own free will, for the sheer enjoyment of it. Clearly, productivity and morale among open-source developers is far superior to that of almost any other team organization model. However, this doesn't mean the Synchronize and Stabilize model is inferior. Microsoft, with their seemingly endless pool of financial resources, can pay employees ridiculous sums of money to do their jobs. On top of that, Synchronize and Stabilize teams stress creativity among team members, allowing developers to design and incorporate their own features and functionality into the product as the project progresses, rather than laying down a strict set of functions to be implemented. Money and creative freedom can do a lot to boost team productivity.

Clearly, the Open Source model is a perfect candidate for comparison to Microsoft's Synchronize and Stabilize model, since successful consumer-level open-source products such as Linux, Netscape Navigator, and Sun's Star Office, can be assessed competitively with Microsoft products such as Windows NT, Internet Explorer, and Microsoft Office. However, Microsoft's model has already been used on a much wider range of products, from operating systems and web browsers, to feature-rich products such as publishing suites and video games. The Open Source model, on the other hand, has only recently gained recognition as a viable team approach, with only a few exceptional products such as Linux. Can Open Source teams, much like Synchronize and Stabilize, be implemented over such a wide range of software products? As mentioned earlier, Open Source development is clearly suited to software products which demand stability and constant refinement, such as operating systems or server software. Open Source teams are also best implemented into projects with highly modular designs, which easily allows smaller, more focused teams to develop naturally [12]. On the other hand, will implementing a Synchronize and Stabilize team guarantee that a company becomes the next Microsoft? This is very unlikely. Microsoft is more than just the Synchronize and Stabilize model [3]. It is an organization made up of extremely well-paid, experienced, multi-talented managers, combined with some of the most intelligent software developers in the world. Rigorous screening processes at the interview level allows the company to handpick only the best developers that properly fit into the Microsoft group ethos [2].

As can be seen, neither of the discussed team models are the most optimal way to organize a development team. The Synchronize and Stabilize model comes close, but falls short in that experimental data from companies other than Microsoft is not available. Microsoft's approach

clearly relies on a handful of other key factors which lead to the success of its Synchronize and Stabilize teams. Similarly, the Open Source model needs to be implemented on a much wider range of consumer-level applications in order to fully assess its adaptability to more than system-level software projects.

9 References

This paper discussed team organization issues related to both Synchronize and Stabilize teams and Open Source teams. A summary of each model's process was described, followed by a comparison of Microsoft's Synchronize and Stabilize teams versus the Mozilla Organization's Open Source teams, as well as each model's strengths and weaknesses. Finally, the applicability of each model to the key requirements of a successful software engineering project were discussed, followed by a brief discussion related to the feasibility of implementing the models in any given project or company. Information was obtained from the following sources:

1. Stephen R. Schach, *Classical and Object-Oriented Software Engineering with UML and Java*, 4th Edition, McGraw-Hill, New York, 1999, pp. 77, 85, 101-102.
2. M. A. Cusamano and R. W. Selby, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, The Free Press/Simon and Schuster, New York, 1995.
3. M. A. Cusamano and R. W. Selby, "How Microsoft Builds Software", *Communications of the ACM* 40 (June 1997), pp. 53-61.
4. J. McCarthy, *Dynamics of Software Development*, Microsoft Press, 1995
5. Linux.org website (<http://www.linux.org>)
6. LinuxHQ.com website (<http://www.linuxhq.com>)
7. Kernel.org website (<http://www.kernel.org>)
8. OpenSource.org website (<http://www.opensource.org>)
9. Josh McHugh, "For the love of Hacking", *Forbes Magazine Online*, (<http://www.forbes.com/forbes/98/0810/6203094a.htm>)
10. Mozilla.org website (<http://www.mozilla.org>)
11. Eric S. Raymond, *The Cathedral and the Bazaar*, (<http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html>)
12. Kim Johnson, *Towards a Descriptive Process for Open-Source Software Development*, (<http://www.cpsc.ucalgary.ca/~johnsonk/SENG/SENG691/towards.htm>)
13. Steve McConnell, "Open-Source Methodology: Ready for Prime Time?", *IEEE Software* (July/August 1999), pp. 6-11
14. Daniel Chudnov, "Open Source Software: The Future of Library Systems?", *Library Journal* (August 1999), pp 40-43
15. Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", *Proceedings of the IFIP Tenth World Computing Conference*. IFIP, 1069-1076
16. Frederick P. Brooks, Jr., *The Mythical Man Month: Essays in Software Engineering*, Addison-Wesley, Reading, MA, 1975.