SWE 4103

Software Quality Assurance and Project Management

Final Report

Concurrent Open Source Software Development

Anthony N. Ilukwe

Electrical and Computer Engineering

University of New Brunswick

Course Leader: Dr. Yevgen Biletskiy

December 8, 2009.

Abstract

This paper examines the *concurrent* software development methodology from the perspective of *open source software development*. The former is the practice of simultaneous software development by a number of developers, while the latter is the process of developing *Open Source* software.

Concurrent open source development is a relatively new development methodology that is still yet to be fully set in stone, and it could pose some challenges, notably change control, to developers. Effective management of projects of this nature is necessary to ensure that these challenges are contained with the application of software engineering techniques.

This paper also discusses reasons why a case can be made for concurrent development; the principles of open source development; the close link between concurrent development and open source development; and tools used in concurrent open source development, with an in-depth look at the critical versioning tool CVS (Concurrent Versioning System).

A scenario, involving two developers working in different parts of the world working on an open source project using CVS for versioning control, is explored to show possible issues that could arise in concurrent software development, and how these issues can be resolved.

Concurrent open source development is then compared and contrasted with other software development methodologies like the *Agile* and *Waterfall* models.

Table of Contents

1. Introduction

Concurrent software development is a software development methodology that involves two or more individuals or teams simultaneously working on a number of development activities within a software project. Concurrent development projects often entail developers who are located in different geographical locations, and this creates the need for these projects to be effectively managed, in order to ensure that productivity is optimal, despite the lack of proximity.

Open Source software is software that is open for users to contribute to, and improve. Open source development is closely tied to concurrent software development, as it involves simultaneous, concurrent, and collaborative development of software.

Concurrent open source development involves a number of developers constantly making modifications. CVS (Concurrent Versioning System) is a tool that facilitates the control and tracking of changes made during concurrent development projects, in order to prevent mishaps such as file over-writing during parallel changes.

Concurrent open source development applies some practices derived from the *Agile* software development methodology, due to the similarities in their underlining philosophies. On the other hand, concurrent open source development has less in common with the traditional software development methodologies, such as the Waterfall model, due to the rigid, less change-accommodating nature of the traditional development cycle. The upcoming sections will examine the above mentioned, as well as other aspects of concurrent open source development, including a scenario in concurrent development project with CVS.

2. Concurrent Software Development

Concurrent development is becoming increasingly popular; with the many advances in communication technology making it easier, faster, and cheaper to communicate and collaborate with people in different parts of the world. The increased popularity of concurrent development can also be attributed to the emergence of Open Source software in the late 90's.

2.1 The Case for Concurrent Software Development

Concurrent software development is a step away from the traditional software development model, due to the fact that is allows a team of developers to work on a software program simultaneously and unite each individual's work at various stages of the project. Some reasons why a case can be made for concurrent development include:

- Supports modularity: Concurrent development supports modularity in software development. For example, in a project that employs the MVC (Model-View-Controller) architecture, each of the three subsystems can be concurrently developed by three different individuals or teams.
- More efficiency in meeting software requirements: Concurrent development often involves several developers, and as a result, there are more heads involved in ensuring that the software meets the requirements. Concurrent development allows for a number of individuals can work on different respective system components; thus resulting in meeting the system goals with higher efficiency.

- Faster delivery of product: The more individuals simultaneously working on a software development project, the quicker the pace of delivery of a finished product. The total amount of time spent on developing software can be significantly shortened by the involvement of more individuals who can contribute their knowledge and skill.
- Keeping pace with new technology: One of the unspoken advantages of concurrent software development is that it gives rise to the possibility of a *smorgasbord* of knowledge and techniques from a variety of individuals involved with a project. This allows for the application of the latest and most up-to-date technology and practices in a development project.
- Keeping pace with rapid software evolution: Having a number of individuals makes it easier to keep pace with maintenance activities that are part of the software evolution process. There is always a high demand for debugging, new features, and upgrades in a software project; concurrent development enables developers to simultaneously make measures that rapidly address the need for continuous, incremental change.

2.2 Open Source Software Development

Open Source software can be described as computer software for which the underlying source code is available for users to use and change to improve the software, and also to redistribute in modified or unmodified forms (Fogel, 2003). Popular open source software include Mozilla Firefox, the LINUX Operating System, and Open Office. Open source

development is often done in a public, collaborative manner, involving several individuals in different locations.

There is a set of principles for open source software that guide the distribution and usage of, as well as, the governing philosophy of open source software. Below is a summary of these principles, in the context of open source software development.

- Availability: Source code must be accessible to everyone and anyone. The source code for open source software is always available for access by whoever wishes.
- Maximal scope of end users: Open source software should be developed without a limited end user scope, as would be the case in commercial software development.
- All developers are equal: There should be no discrimination among developers, due to any criteria. Everyone involved in an Open source project is of equal status.
- Open source implementation can be offered as extensions or subsets.
- Collaboration and concurrent development must be done to continually improve the software product.
- There is freedom to create good product without external pressure: Unlike a standard commercial software development project, there are no external factors such as marketing strategy and budgeting to deal with. Developers can focus solely on creating high quality software.

It is easy to see the intersection between concurrent software development and open source development. Open source software development employs concurrent development, typically involving several people simultaneously working on the same software development project, and possibly the same source code. The two concepts

essentially go hand-in-hand. Because lack of regular face-to-face communication among developers could pose a challenge in these projects, there are a range tools used for communication (Instant messaging, mailing lists, Wikis), bug tracking (BugZilla, Scarab), and versioning control. The latter will be discussed in more detail in the next section.

3. CVS in Concurrent Open Source Software Development

Concurrent Versioning System, commonly known as CVS, is a system developed to facilitate the tracking of changes made to a set of files. CVS is used to control modifications made to source code, and it saves the entire revision history of a set of source files in a repository, which is a file tree saved on a central remote server. Additionally, CVS allows users to roll back to a previous state saved in its revision history.

CVS is highly vital to the collaborative nature of concurrent software development, because it allows files to be simultaneously modified by any number of developers who have access to the repository.

Unlike its predecessors, such as RCS (Revision Control System), CVS does not have a Lock-Modify-Unlock mechanism in which the repository is locked while only one developer can work on source code, after which it can then be unlocked. On the other hand, CVS has a Copy-Modify-Merge mechanism that allows multiple developers to checkout a copy of a file(s) from the repository to their local machine. This copy can then be checked in to the repository after modification. CVS provides features that ensure there are no conflicts or overlaps in the code checked in by different developers. CVS can be used through a command line or terminal interface. Popular IDEs (Integrated Development Environments) such as *Eclipse* and *Netbeans*

have a CVS plugin that allows developers to use CVS from within their local development environment.

3.1 CVS Terminology

Below is a brief description of some of the main terminologies used in CVS.

- **Working Copy:** This is the copy of the software project that exists on the local machine of an individual developer.
- **Master Copy:** This is the most recent software copy that is saved on the remote server's repository.
- **'Update':** This is the migration of modifications on the master copy to the working copy. It ensures that developers' working copy is in synch with the most current revision of the code in the repository.
- **'Checkout':** This is the process of making a request for a local working copy of the project from the repository.
- **'Commit':** This is the process of sending modifications from the local working copy to the project's remote repository.
- **'Diff':** This is a feature of CVS that allows a developer to view the changes that have been made to a given file, and comparatively analyze the difference(s) between the file and past revision of the file.
- **Trunk**: This the base of the development project in CVS, and the location where the master copy of the project is contained.

- **Branch:** This is an offshoot of the trunk that enables developers to create a renamed duplicate of the trunk, which can be modified in parallel and then joined or 'merged' back to the main trunk.
- **'Merge':** This is the process of joining a branched and modified revision of the trunk back to the main trunk.

3.2 A Scenario in Concurrent Open Source Software Development with CVS

As mentioned in the preceding section, CVS facilitates the tracking and control of changes made within a software development project. Some CVS features, such as **merge** and **commit**, prevent situations like the over-writing of parallel changes; thus minimizing human error. The below scenario shows a real-world illustration of how CVS can prevent possible havoc within a concurrent development project.

Scenario: There are eight developers working on an open source project to develop a new plugin for the latest Mozilla Firefox browser. This team also includes two freelance developers; *Franz* in Frankfurt, Germany and *Sally* in Sydney, Australia. The ten-hour time difference means that very often, Franz is asleep while Sally is busy at work, and vice-versa.

- (8pm Frankfurt, 6am Sydney) Franz checks outs a working copy to his local machine and begins to make modifications to the code.
- (11pm Frankfurt, 9am Sydney) Franz saves his work and closes for the day. He does not commit his code modifications.
- (12am Frankfurt, 10am Sydney) Sally checks out a working copy, and begins to make modifications, unaware that Franz has already begun work on the same code.

- (6am Frankfurt, 4pm Sydney) Sally completes her work and commits it to the repository.
- (11am Frankfurt, 9pm Sydney) Franz, unaware that Sally has modified the master copy, continues to work on his local copy, and finally saves, then tries to commit.

CVS does not allow the commit operation to take place, due to the conflict it detects. There are differences between Franz's working copy and the revision that was committed by Sally just a few hours earlier.

Utilizing a command/feature called **diff**, CVS allows Franz to examine the differences between his working copy and Sally's copy, which is now the master copy. Figure 1 shows what Franz sees using the **diff** view.

a)

```
1 Method add(int a, int b) {
2      a + b;
3 }
4 //last modified by Sally
```

b)

```
1 Method add(int a, int b) {
2      b + a;
3 }
4
5 Print(add(6,7));
6 //last modified by Franz
```

Figure 1: CVS Diff view showing snippets of Franz's and Sally's code with the conflicting line 2 italicized.

In the code snippet in Figure 1(b), CVS allows lines 5 and 6 to commit; however, it detects a conflict on line 2, and Franz has to modify the second line in his working copy to match with

Sally's in Figure 1(a). The **diff** view allows Franz to see the differences between his working copy and the master copy, which is also Sally's version. As soon as Franz updates his second line to match Sally's, as seen in Figure 2, CVS allows him to commit his revision.

```
1 Method add(int a, int b) {
2       b + a;
3 }
4
5 Print(add(6,7));
6 //last modified by Franz
```

Figure 2: Franz's code, after he modifies line 2. CVS will permit this to commit.

CVS helped prevent a potential code breakage; however, the scenario could have been avoided by:

(a) Better communication: Franz and Sally could have exchanged emails to inform each other about the status of the work they were doing.

(b) Branching: Franz and Sally could have created **braches**, allowing for their work to be done in parallel, but branched out of the main trunk. Consequently, either Franz's or Sally's branch can be merged back into the main trunk when deemed appropriate.

4. Comparison with Other Development Methodologies.

Concurrent software development is a non-traditional software development methodology that has some significant differences in its approach, compared to that of other common software

development methodologies like the *Agile* and *Waterfall* models. However, it also shares some similarities with these software development methodologies.

I. Organizational Structure: In concurrent open source development, there is no official project manager, as would be the case in a traditional development setting. Every developer within the project is of equal status, although the initiator(s) may indirectly bear more responsibility for the completion of the project.

II. Development Structure: Concurrent open source development often omits the elicitation of requirements at the start of the project, but rather, bases the gathering of requirements on early releases of the product (Robbins, 2003, p. 6). Agile development practices such as *Extreme Programming* and *Internet Speed Development* can be conveniently applied to concurrent open source projects due to their flexibility, which is necessary for projects of this nature. The incremental and iterative style involved with agile development is well suited for concurrent open source development.

Running concurrent open source projects using the more traditional methods, such as the Waterfall model, could be challenging due to the fact that the latter is plan-driven and does not accommodate going back to previous phases.

III. Proximity and Communication: Most concurrent open source projects involve people in different geographical locations; often in different countries and time zones. Most traditional development projects, on the other hand, typically involve teams of developers within the same organization and in close proximity to each other.

IV. Developers: The developers involved Concurrent Open Source projects usually have full-time careers or other jobs, and work on these part-time projects during their personal spare time. According to Feller, Fitzgerald, Hissam, & Likhani (2005) most of these

developers are motivated by enjoyment and the desire to develop their skills and human capital.

This is quite different from a traditional development setting, in which the developers are usually full-time paid employees. Consequently, in concurrent Open Source projects, there is a higher demand for individuals who are motivated, self-organizing, and effective communicators.

5. Conclusion

Concurrent Open Source development is a relatively new software development methodology that could pose a challenge to software developers; however, the use of versioning tools like CVS, and effective communication mechanisms ensure that software engineering practices are applied to improve the efficiency of the development process.

Versioning tools help control and manage changes made to the source code in a project, and each revision is stored in a tree on the server repository. After a set of **revisions** and **releases** have been created, debugged, and validated by other members of the Open Source community, the project team can jointly decide to release a new bundled **version** of the software.

In addition to CVS, which was discussed in this paper, there are other similar versioning systems, such as *Subversion* and *Git,* which are commonly used by developers in concurrent development projects.

References

Feller, J., Fitzgerald, B., Hissam, Scott., Likhani, K.R. (2005). *Perspectives on Free and Open Source Software*. Cambridge, Massachusetts: The MIT Press.

Fogel, K. (2003). *Open Source Development with CVS*, 3rd ed. Scottsdale, Arizona: Paralyph Press Inc.

Robbins, J. E. (2003). *Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools.* Retrieved from http://www.ics.uci.edu/~wscacchi/Papers/New/Robbins-msotb-OSSE-Aug03.pdf