Adopting OSS Methods by Adopting OSS Tools

Jason E. Robbins
CollabNet, Inc.
8000 Marina Blvd., Suite 600
Brisbane, CA 94005-1865
(650) 228 2539
jrobbins@collab.net

ABSTRACT
The open source movement has created and used a set of software engineering tools with features that fit the characteristics of open source development processes. To a large extent, the open source culture and methodology are conveyed to new developers via the toolset itself, and through the demonstrated usage of these tools on existing projects. The rapid and wide adoption of open source tools stands in stark contrast to the difficulties encountered in adopting traditional CASE tools. This paper explores the characteristics that make these tools adoptable and how adopting them may influence software development processes.

General Terms
Management, Design, Standardization, Verification.

Keywords
Open source software engineering, methodology adoption.

1. INTRODUCTION
One of the biggest challenges facing the software engineering profession is the need for average practitioners to adopt powerful software engineering tools and methods. Starting with the emergence of software engineering as a field of research, increasingly advanced tools have been developed to attempt to address the difficulties of software development. Often these tools addressed accidental [2] difficulties of development, but some have been aimed at essential difficulties such as management of complexity, communication, visibility, and changeability. In the 1990's, the emphasis shifted from individual tools to the development process in which the tools were used. The software process movement produced good results for several leading organizations, but it did not have much impact on average practitioners.

Why have CASE tools not been used? Often the reason has been that they do not fit the day-to-day needs of the developers who were expected to use them: they were difficult to use, expensive, and special purpose. The fact that they were expensive and licensed on a per-seat basis caused many organizations to only buy a few seats, thus preventing other members of the development team from accessing the tools and documents and reducing the impact of the tools on the project. One study of CASE tool adoption found that adoption correlates negatively with end-user choice, and concludes that successful introduction of CASE tools must be a top-down decision from upper management [3]. The result of this approach has repeatedly been shelfware: software tools that are purchased but not used.

Why have advanced methodologies not been widely adopted? Software process improvement efforts built around CMM or ISO-9000 have required resources normally only found in larger organizations: a software process improvement group, time for training, outside consultants, and the willingness to add overhead to the development process in exchange for risk reduction. Top-down process improvement initiatives have often resulted in a different kind of shelfware where thick binders describing the organization's software development method go unused. Smaller organizations and projects on shorter development cycles have often opted to continue with their current processes or adopt a few practices of lightweight methods such as extreme programming [1] in a bottom-up mannor.

In contrast, open source projects are rapidly adopting common expectations for software engineering tool support and those expectations are increasing. Just three years ago, the normal set of tools for an open source project consisted of just a mailing list, a known bugs list, an INSTALL text file, and a CVS server. Now, open source projects are commonly using tools for issue tracking, source code management, design and code generation, automated testing, and packaging and deployment. The featuresets of these tools are aimed at some key practices of the open source method, and in adopting the tools, software developers are predisposed to also adopt open source practices.

Exploring and encouraging development and adoption of open source software engineering tools has been the goal of the tigris.org web site for the last two years. The site hosts open source projects that are developing software engineering tools. Tigris.org also hosts student projects on any topic, and a reading group for software engineering research papers. The name "Tigris" can be interpreted as a reference to the Fertile Crescent valley between the Tigris and Euphrates rivers. The reference is based on the hypothesis that an agrarian civilization would and did arise first in the location best suited for it. In other words, the environment helps define the society, and more specifically, the tools help define the method. This is similar to McLuhan's proposition that "the media is the message" [4].

2. SOME APSECTS OF OSS AND OSSE
The open source movement is broad and diverse, so it is difficult to make generalizations; however, there are several common aspects that can be found in many open source software products

ICSE '02, May 25, 2002, Orlando, FL.

and projects, and in the open source software tools used to produce them.

- **Universal, immediate access to project artifacts**. The heart of the open source method is the fact that the program source code is accessible to all project participants. Beyond the source code itself, open source projects tend to allow open access to all software development artifacts such as requirements, design, open issues, rationale, development team responsibilities and schedules. Access to this information is not just allowed, it is typically available in real-time over the Internet. This means that all participants can base their work on up-to-date information.
- **Volunteer effort**. Open source projects typically have only very few dedicated staff. Instead, work is done by developers who volunteer their contributions. This means that every feature is validated by at least on person who strongly desires it. Another aspect of volunteerism is that unlikable jobs tend to go undone, unless they are automated.
- **Standards-based**. Lack of formal requirements generation in specific open source projects tends to force reliance on external standards or conventions. Deviation from standards is discouraged because of the difficulty of specifying the alternative with the same level of formality and agreement among contributors. Standards also define interfaces that give choice to users and support diversity of usage.
- **Diversity of usage leads to plurality of authorship**. Open source products are often cross-platform from the start, and usually offer a wide range of configuration options that allow them to address diverse use cases. When contributors add a feature to "scratch an itch," it can lead to feature creep and a loss of conceptual integrity [2, 5]. This can make it harder to meet predefined deadlines, but it broadens the appeal of the product because more potential users get their own win conditions satisfied. Peer review and limited resources can help keep a lid on feature creep.
- **Release early, release often**. Open source projects are usually not subject to the economic concerns or contractual agreements that turn releases into major events in traditional development. In fact, open source projects need to release pre-1.0 versions in order to build the development community needed to reach 1.0. The feedback provided by this practice is one key to risk management in open source and is also found in other methods.
- **Peer review**. Feedback from users and developers is one of the practices most central to the open source method [5]. Peer review has also been shown to be one of the most effective ways to eliminate defects in code regardless of methodology.

3. SOME COMMON OSSE TOOLS

This section reviews several open source software engineering tools with respect to the aspects defined above. Most of these tools are already widely used, while a few are not yet widely used but are rapidly expanding their user base.

3.1 Version Control

CVS. The concurrent versions system (see cvshome.org) is the most widely used VC system in open source projects. Its features include: a central server that always contains the latest versions and makes them accessible to anyone over the Internet, with support for disconnected use; conflict resolution via merging rather than locking to reduce the need for centralized coordination among developers; simple commands for checking in and out that lower barriers to casual usage; and, cross-platform clients and servers. It is common for CVS to be configured to send email notifications of commits to project members to prompt peer review.

Subversion. Subversion is being developed as the official successor to CVS (see subversion.tigris.org). Its featureset includes essentially all of CVS's features, but it is also based on the existing WebDAV standard (see webdav.org), and has stronger support for disconnected use.

3.2 Issue Tracking and Technical Support

Bugzilla. Bugzilla was developed to fit the needs of the Mozilla open source project (see bugzilla.mozilla.org). Its features include: an "unconfirmed" defect report state needed for casual reporters who may not share knowledge of previous issues; a "whine" feature to remind developers of issues assigned to them, this addresses the lack of traditional management incentives; and, web-based interface that makes the tool cross-platform, universally accessible, and that lowers barriers to casual use.

Scarab. The scarab project, much like subversion, seeks to establish a new foundation for issue tracking systems that can gracefully evolve to fit many needs over time (see scarab.tigris.org). Key features of scarab include many similar to those of Bugzilla, plus: issue de-duplication on entry to defend against duplicates entered by casual participants; standard XML issue-exchange formats; and, highly customizable issue types, attributes, and reports.

3.3 Technical Discussions and Rationale

Mailing lists. Mailing lists provide a key feature above simple direct email in that they typically build archives that capture the design and implementation rationale. Since mailing lists are based on email, they are standards-based, cross-platform, and accessible to casual users. Also, since the email messages are free-format text, this single tool can serve a very wide range of use cases, although it relies on social conventions to provide much specific support for any particular use case.

FAQs and FAQ-o-matic. Lists of frequently asked questions help to mitigate two of the main problems of mailing lists: the difficulty of summarizing the discussion that has gone before, and the wasted effort of periodically revisiting the same topics as new participants join the project. FAQ-o-matic and similar tools aim to reduce the unlikable effort of maintaining the FAQ.

3.4 Build Systems

Make. The unix 'make' command is a standard tool to automate the compilation of source code trees. Its usage in open source is an example of the use of automation to reduce barriers to casual contributors. Not only are 'make' and associated tools used in

open source projects, but there are several conventions that make it easier for casual contributors to deal with different projects.

Ant. Ant is a java replacement for 'make' that uses XML files instead of makefiles (see jakarta.apache.org). Ant's use of the XML standard is a key enabler of reuse of standard XML parsing libraries, and a key to data exchange with other tools to support integration into a variety of IDEs. Ant's concept of a task is at a higher level of granularity than a command line in a makefile; this can reduce the tedium of managing complex makefiles, increase consistency across projects, and ease peer review.

3.5 Design and Code Generation

ArgoUML. ArgoUML is a pure-java UML design tool (see argouml.tigris.org). In addition to being cross-platform and standards-based, it emphasizes ease of use and goes so far as to actively help train casual users in the usage of UML.

Torque. Torque is a java tool that generates SQL and java code to build and access a database defined by an XML specification of a data model (see jakarta.apache.org). It is cross-platform, customizable, and standards-based. Torque's code generation is customizable because it is template-based; also, a library of templates has been developed to address incompatibilities between SQL databases. Together these features can greatly reduce the unlikable task of making the adjustments needed for new products to support multiple databases.

3.6 Integrated Development Environments

Emacs. Emacs is "the extensible self-documenting text editor" (see savannah.gnu.org). Its name alone emphasizes that it serves diverse use cases and attempts to lower barriers to casual users. It is also cross-platform and includes bug-reporting tools supporting the release-early-release-often practice.

NetBeans. NetBeans is a pure java IDE for java development with a very well thought-out framework for integrating new modules (see netbeans.org). Its ability to add modules supports diverse usage, and there are several example of modules that aim at supporting standard open source tools such as CVS and Ant.

Eclipse. Eclipse is a recent open source IDE effort that has many of the same goals as NetBeans, but seeks to serve an even more diverse userbase by supporting multiple languages (see eclipse.org).

3.7 Testing Tools

Junit. Junit supports java unit testing (see junit.org). It is a simple framework that is highly customizable. Its aim is to reduce the unlikable task of manual testing.

Cactus. Cactus is an extension to the Junit framework that uses Ant to easily automate testing of server-side web applications (see jakarta.apache.org).

3.8 Packaging and Deployment

RPM. The Redhat Package Manager (see rpm.org) is a system for packaging and deploying software components on Linux and other operating systems. It has a simple command syntax for its most basic operations, allowing casual usage. It aims to eliminate the unlikable task of building installers and maintaining hosts.

3.9 Missing Tools

Although there is a wide range of open source software engineering tools available to support many software engineering activities, there are also many traditional development activities that are not well supported. For example, requirements management, project management, metrics, estimation, scheduling, program analysis, and test suite design.

4. IMPACT OF ADOPTING OSSE TOOLS

Drawing conclusions about exactly how usage of these tools impacts development methods in practice would require careful observation of actual projects. The descriptions above can help guide such observation to look for the following benefits:

- Since the tools are free and support casual use, more members of the development team will be able to access and contribute to artifacts in all phases of development.
- Since the "source" to all artifacts is available and up-to-date, there is less wasted effort due to decisions based on out-of-date information.
- Since causal contributors are supported in the development process, non-developer stakeholders such as management, sales, marketing, and support, should be more able to constructively participate in the project.
- Since many of the tools support incremental releases, teams using them should be better able to release early and often.
- Since many of the tools aim to reduce unlikable work, more development effort should be freed for forward progress.
- Since peer review is addressed by many of the tools, projects may be able to catch more defects in review or conduct more frequent small reviews in reaction to changes.

5. REFERENCES

[1] Beck, K. Extreme Programming Explained – Embrace Change. Addison Wesley Longman, Reading MA, 2000.

[2] Brooks, F. P. No silver bullet: essence and accidents of software engineering. IEEE Computer, 20, 4, 10-19.

[3] Iivari, J. Why are CASE tools not used? Commun. ACM 30, 10, 94-103.

[4] McLuhan, M. Understanding Media. MIT Press, Boston MA, 1994.

[5] Raymond, E. S. The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly and Associates, Sebastopol CA, 2001.