Beyond Code: Content Management and the Open Source Development Portal (Position Paper)

T. J. Halloran William L. Scherlis
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{thallora|wls}@cs.cmu.edu

Justin R. Erenkrantz
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
jerenkra@ics.uci.edu

1 Introduction

Open source project collaboration web portals (e.g., Mozilla.org, SourceForge.net) have become the focal point for interaction with and development of most open source software projects. These collaboration portals allow considerable community interaction with a project while respecting and maintaining effective control by the project's leaders over process, architecture, participation, and quality. A variety of successful and widely used open source collaboration tools have evolved (e.g., CVS, Bugzilla, Mailman) specifically to support tool mediation of this interaction. However, use of a *Content Management System* (CMS) beyond simply storing the web site's contents alongside the project's code within CVS is rare. Is there a real need for a CMS within an open source project portal? What attributes does a CMS system need to be adoptable? Is more ambitious CMS adoption consistent with open source practice?

We take the position that an advanced CMS *is* a useful addition to an open source development portal and is consistent with the current trajectory of open source collaboration tool evolution. However, to be adoptable a CMS must respect and follow open source practice—not try to redefine it—and facilitate an incremental transition from existing content management methods. In addition, as open source practice is clearly not homogeneous, a "one-size-fits-all" CMS will not be successful. As a project's community grows to include individuals with strong supporting roles other than programming (e.g., documentation, translation, QA) the need for a CMS becomes more acute.

What can a CMS do for an open source project? It can aid project information awareness, assist project/personal workflow, and facilitate the use and maintenance of models. These are significant functions that can help projects express/exploit more information regarding design intent, provide better degrees of assurance for code safety/dependability, and perhaps even facilitate a more agile approach to structural/architectural change.

2 Roles and benefits

Content management refers to the management of web site content. We loosely define a CMS to be any automated tool designed to support the content management function. Hence, the common practice of storing the project's web site contents within CVS alongside a project's source code would qualify—but only at the functional low-end. A high-end CMS typically assists users with web site authoring (e.g., directly within a browser or via more specialized tools), document organization, workflow, multi-format publishing (e.g., HTML, WAP, PDF), version control, archiving, and security.

What role would a CMS fill in an open source development portal? To answer this question we first need to identify the information managed by an open source portal. There are generally five notional content databases:

Database	Content (*Typical Tool Support*)
Source Code	The project's source code/versions/logs (*CVS*)
Bugs/Issues	Project defect/enhancement reports (*Bugzilla, GNATS*)
Discussion	Mailing list/newsgroup archives (*Mailman, Google Groups*)
Testing	Nightly build/regression results (*Tinderbox, Mailman*)
Documentation	Documentation, process/workflow, marketing, community/developer information (*CVS*)

It is within the *Documentation* database/content area that a CMS can provide the most immediate utility to an open source portal. We include under this rubric informal documentation, formal documentation (such as low-level models), and more aggressive approaches to linking and consistency management among these diverse information assets. Note the other four database/content areas have had specialized tools evolve to support them—CVS, specifically developed for source code control, has really only provided an expedient stop-gap solution for CMS functionality. We believe a more ambitious CMS than CVS alone provides a better overall portal solution (even if the CMS stores its low-level content in CVS) and will provide long-term benefits in the following areas:

- *Information/context awareness:* How can a developer/participant restore awareness in project activity after having been "offline" for a few hours, days, or weeks? Present approaches include browsing CVS commit-logs, "my bugs" in Bugzilla, and other ad hoc approaches learned over time to be effective by the project's community. A CMS augments existing ad hoc approaches by increasing awareness in the *Documentation* database/content area discussed above. In addition, the CMS should include a capability for developers to tailor what areas of the project they are interested in. There are the usual trade-offs between extent of tailoring to individual needs and the extent of up-front configuration effort required to achieve this.

- *Process support:* How can a project better institutionalize workflow support without adding a "bureaucratic" burden to the developers? This is a question often asked by industry software managers considering adopting open source engineering methods and tools. Present practices include informal coordination (e.g., the "CHANGES" file under CVS used to help coordinate work among Apache HTTP Server developers) and use of bug/issue databases (e.g., the use of Bugzilla as a project management tool by the Mozilla programmers—loading milestones such as "Ship Mozilla 1.0" as "bugs"). A CMS adds a workflow infrastructure to the open source portal. Workflow automation streamlines processes that are difficult today (e.g., routing proposed web page changes by non-committers to committers, notifying all developers that have recently checked-in changes to a group of code that its documentation has been updated, tracking and communicating workflow progress to project leaders).

- *Individual process support:* How can an individual developer use awareness information and associated CMS tool support to manage priority setting in development effort and synchronization points with other developers? Present approaches are informal, including use of general mailing lists, bug-specific "mailing lists", and instant messaging. This is *not* related to PSP, TSP, etc., which focus on metrics use and optimization of productivity and quality. The focus here is on allocation, prioritization, and timing of effort.

- *Use of models:* Open source engineering practices focus around code, and, in present practice, there appears to be a pragmatic constraint that any use of models must derive from the "ground truth" of code. How can models be created and managed to support the expression of design intent not directly manifest in code? In particular, how can models be linked with code in a direct tool-managed manner to support consistency management, analysis, and other model-related functions—we believe a CMS could facilitate, to some degree, this type of capability. An example of a kind of "model" that fits this approach is the use of scaffold and unit tests incorporated into a build (e.g., the use of JUnit on the Eclipse project). A more forward looking example is assurance that a model of the code's concurrency policy is consistent with the project's source code [3, 8].

Our notion of what constitutes a CMS is broad and inclusive. Commercial CMS products range from large high-end enterprise systems, such as Interwoven, to low-end systems, such as Microsoft FrontPage, with many products in between. Of more interest to us, however, are the many open source CMS projects under active development today. At the low-end, Wiki allows web page visitors to directly edit page content within their web browser. There are more than 70 active open source Wiki projects. The various Wiki alternatives support varying degrees of CMS functionality. For example, some allow anyone to change site content while others require authentication, some use a version control system to archive and track changes others do not, and so on. A few open source CMS projects with grander aims than Wiki functionality exist as well. These projects, which include Zope and OpenCms, contain functionality similar to mid- to high-end commercial CMS. Which approach is best for an open source portal? We will return to this question after examining current open source portal CMS experiences.

3 Direct experience in our research projects

Our interest in CMS/open source portal integration evolved from two experiences setting up, using, and maintaining open source-style development portals. The first development portal was used by our research group (with 15 team members dispersed between Carnegie Mellon University and the University of Wisconsin–Milwaukee) to develop 160KSLOC of Java software. We grew to rely heav-

ily upon the open source tools and found them adequate—except in one area: web content management. Within the last year alone we changed from managing content in RCS, to using no revision control at all, to using CVS, to augmenting CVS with some CGI/Python publishing scripts, and finally to a site based on the Plone CMS.

The second portal, which was setup much more recently, supports collaboration within the High Dependability Computing Program (HDCP) on an aggressive software development using the Real-Time Specification for Java for a NASA project. This open source-style web portal supports 10 researchers and practitioners from Carnegie Mellon University, Carnegie Mellon-West, Caltech's Jet Propulsion Labs, and Sun Microsystems. We installed a Wiki as the main page for this portal to avoid the content management problems encountered in our earlier research portal. No problems have arisen and the Wiki has been popular.

Both these projects are closed groups with little public interaction. However, these experiences raised our interest in the role a CMS can play in a open source portal.

4 Experience in the open source community

To better understand the challenges associated with CMS/open source portal integration, we informally surveyed several open source portals. In this section we briefly report on a few interesting cases of attempts to integrate some form of CMS within a real-world open source portal.

4.1 SourceForge.net

SourceForge.net is perhaps the best known-open source web portal in the world. As of February 2003 it hosted over 56K projects and had 565K registered users. Several successful and well-known open source projects are hosted on SourceForge.net (e.g., Python, JBoss, MySQL) but, due to no barrier to entry except use of an open source license, lots of "dead" projects "haunt" this site as well.

SourceForge.net provides each project a directory in which to place its web content. The `scp` (secure copy) command is used to upload web content to SourceForge.net. This is the most primitive approach we encountered—all content management must be setup and handled by the development team for each SourceForge.net project or hosted at another server. In practice, many SourceForge.net projects store their web content within their project's source code CVS repository and use a simple script to publish it.

4.2 Mozilla.org

Mozilla.org is the web portal for the development of the Mozilla web browser. The highly publicized Mozilla open source project was started in 1998 by Netscape and included the creation of the Mozilla.org collaboration site. In June 2002 the Mozilla project reached a major project milestone by releasing Mozilla version 1.0. Mozilla.org has also contributed several widely used open source collaboration tools such as Bugzilla (issue tracking and project management) and Tinderbox (portability and regression testing).

The Mozilla.org web site has used and is still using CVS for management of site content not managed within other tools such as Bugzilla. An approach to general project documentation and web site content management has been under debate within the Mozilla project for a long time. As a post to several Mozilla newsgroups noted in December 2000, "It's a very "big" problem (336Mb, 30,716 files)" [6]. Long newsgroup threads debating the merits of various approaches appear several times since 2000 on the Mozilla documentation newsgroup. The real impact of this problem is that some volunteers to work on Mozilla documentation were lost—CVS, at least alone, was not succeeding as a viable CMS for the Mozilla project.

In August 2002 the existing problems were summarized by Mitchell Baker, Mozilla's Chief Lizard Wrangler, as "Once nice docs exist, it's hard to get them to [Mozilla.org]," specifically: (1) "learning CVS is a burden," (2) "using CVS is awkward," (3) "finding a [document's] location is difficult/impossible due to current poor organization," and (4) "maintaining the pages is time-consuming." Baker also notes, "I understand the desire to set up an over-arching structure for everyone to solve the problems. But this is not the approach which prospers in the rest of the project, our rules and structures have grown incrementally" [2].

In December 2002 a simple CMS called *Doctor* was added to help manage site content. Mozilla.org's Doctor system adds an "Edit this page" link to the bottom of each web page—it is essentially a Wiki with access control. Doctor is a wrapper around the CVS document management system already in use that allows in-browser edits of a web page's HTML content. Doctor protects against defacement of Mozilla.org by requiring a valid CVS identification and password to publish any change. However, Doctor lacks any built-in workflow capability. You are not allowed to simply route a suggested website change to a known project committer within this tool—you must create a Bugzilla bug to suggest your change.

Mozilla is experimenting with a Zope/Plone site which is hosted at `moz.zope.org`. Zope is full-featured open source CMS. Plone is built on top of Zope and provides more "out-of-the-box" capability than Zope alone. This site is the only use (although still in trial) of a high-end CMS by a major open source portal we encountered in our preliminary investigation (with the logical exceptions of projects like OpenCms, Zope, and Plone). Success for the Mozilla project with this approach is not a foregone conclusion.

Newsgroup postings note many limitations and bugs with the experimental site—but good relations appear to exist between these three open source projects and steady progress is being made.

4.3 PHP

The web portal for the development of PHP, a server-side cross-platform HTML embedded scripting language, allows a series of *user contributed notes* to be attached to the official documentation pages about the PHP system. These notes can be contributed by any site visitor and appear at the bottom of the documentation page. User contributed notes are a popular addition to the documentation as illustrated by the below mailing list quote, one of two that were posted, defending them from a advanced PHP user claiming they were not useful to him:

> I hope you meant they are outdated in some parts. Because, the user notes are very very useful for tons of people. It 1) suggests a function's usage 2) extends the documentation (often [there are bugs in what gets] into the official description). Though a cleanup would be good [4].

Note that this content management capability allows for any user to contribute simple items directly into the documentation with very little effort—without allowing the official documentation to be changed. Members of the official PHP documentation team can use these notes to subsequently improve the official documentation. PHP's user contributed notes system is a good example of a simple CMS that has been successfully incorporated into an open source portal.

4.4 Apache Software Foundation

The Apache Software Foundation (ASF) is a highly decentralized community of developers supporting 17 major projects (many with sub-projects). In December 2002 the ASF started using a Wiki hosted at `nagoya.apache.org` for some of its projects, including the well-known HTTP Server and Jakarta projects. Several concerns arose that generated significant discussion among the ASF community. One of the authors of this paper, as an ASF member, was involved in these discussions.

The initial Wiki had no ability to provide notifications of content changes. This made it difficult to maintain an awareness of changes to the site's content over time. As an attempt to address this concern, the Apache Wiki had Rich Site Summary (RSS) support added to it. RSS facilitates a weblog-like (pull-based) notification of Wiki changes. This approach was not popular with Apache developers because "while email is a generally used tool around the ASF, weblog and related technologies are not as common" [7]. Hence, the community decided that push-based email notifications were a better fit for the ASF and the Wiki was again modified to send "change-emails" to an archived mailing list.

Another more serious concern with the fledgling Wiki was maintaining oversight. First, because the Wiki was hosted on an ASF computer it raised some liability concerns or as one developer put it, "oversight of the type that the ASF as a US incorporated is supposed to maintain" [9]. Second, since the ASF is a collection of communities rather than a single project sharing the same Wiki complicates content oversight because no single project community can do it. The foundation has an existing organizational structure in its Project Management Committees that ensure oversight over the code and traditional websites [1]. However, as the below mailing list quote illustrates the Apache Wiki has raised several new policy questions:

> My concern is over where do we draw the line—after the oversight is in place. The extremes are clear—porn will be removed, and excellent documentation will be included in the products and their authors may become committers.
>
> What happens in between is a different story. My opinion is that Wiki should be treated as mailing lists—and not as source code in CVS and subject to consensus.
>
> The real problem is not the warez or porn—that's something we'll know how to handle. What if someone creates a page Apache*Foo*Sucks (where *Foo* is one of the Apache projects)? And it includes a list of problems and arguments—just like he would do it in the mailing list. Are we going to remove it—or just create Apache*Foo*IsGreat with counter-arguments? What if it's about JCP? Or GPL? Or the best web development technology? Do we keep or remove those pages? [5]

Very recently, a proposal to split the Apache Wiki into realms of oversight that map better to individual Apache projects and sub-projects has been made. Today, however, the original Apache Wiki is still in use.

5 CMS requirements

Based upon our experiences we believe the following list of requirements should be considered to help ensure successful integration/adoption of any CMS as part of an open source development portal:

- *Fit-in with established portal tools:* A CMS designed to be integrated within an open source portal must cooperate with the well established tools. Its role is not

to replace the project's mailing lists or to do away with bug/issue tracking tools. In addition, the CMS must allow incremental transition from existing practice on the portal.

- *Assist, don't burden, the project leaders with oversight:* The CMS should allow the leaders of the project to exercise fine-grain control over the abilities of each and every registered user. The web site content should be able to be divided into sections, including a hierarchy of sub-sections, to ensure that permissions for a user are not all or nothing.

- *Facilitate contributions:* The CMS should allow a project to lower the barrier to entry for someone wanting to contribute. Some examples include allowing web page editing directly within a browser (e.g., Wiki or Mozilla's Doctor) even in a limited and controlled manner (e.g., PHP's user contributed notes). This is important because most project portals offer resources to help potential new participants quickly reach the point of becoming visible and acknowledged contributors to the project.

- *Facilitate awareness:* The CMS should facilitate keeping project members aware of ongoing changes within the project. This capability should allow individuals to "tune" their interest about project activities to avoid information overflow. Push as well as pull change notification should be supported.

- *Support workflow:* The CMS should add an infrastructure for workflow within an open source portal. This capability could be used to facilitate further integration between the portals notional databases/content types. A simple example of workflow would be to route a proposed web page change to a project focal-point who can then review the change and accept it or reject it.

- *Facilitate content organization and models:* The CMS should assist users with document/content organization. This capability must not be all-or-nothing to facilitate incremental adoption of site organization. In addition, facilitating models of design intent linked to the project's code would allow the portal to "get more semantic."

- *Archiving and metrics*: The CMS should remember all committed, as well as rejected, changes to the site. CMS use of the same system used to control source code would help to simplify site maintenance (e.g., CVS, Subversion). A general tabulation of statistics about changes and who submitted them when should be kept (to allow a project to spot trolls or scripts systematically submitting bogus change requests).

6 Summary

We have presented the position that a CMS can fulfill a useful role within an open source development portal and reported on some limited CMS experiences within the open source community. We hypothesize that successful open source tool adoptions are characterized by a Principle of Early Gratification—that increments of investment by project participants must be very closely followed by increments of return on that investment. This Principle provides useful design guidance for a CMS. It is all too easy, especially with highly-visible "one-size-fits-all" portal solutions like SourceForge.net, to view open source portal capabilities and tools as well understood and static—in reality these portals are under constant evolution, driven by evolving project needs. We believe that more ambitious automated content management is evolving into a useful and accepted piece of the open source development portal.

References

[1] Apache HTTP Server Project Guidelines. `http://httpd.apache.org/dev/guidelines.html`. Current Feb. 2003.

[2] M. Baker. (Mozilla.org) Documentation effort. `http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&safe=off&selm=3D63ADEF.8070700%40mozilla.org`. Current Feb. 2003.

[3] A. Greenhouse and W. L. Scherlis. Assuring and evolving concurrent programs: Annotations and policy. In *Proceedings of the 24th International Conference on Software Engineering*, pages 453–463, New York, May 2002. ACM Press.

[4] M. Maletsky. (PHP.net) re: [php-doc] re: Php documentation authors / editors and license. `http://marc.theaimsgroup.com/?l=phpdoc&m=104421737507766&w=2`. Current Feb. 2003.

[5] C. Manolache. (Apache.org) Wiki - we have a problem :). `http://nagoya.apache.org/eyebrowse/ReadMsg?listName=community@apache.org&msgNo=1353`. Current Feb. 2003.

[6] G. Markham. (Mozilla.org) Website reorganisation. `http://groups.google.com/groups?q=g:thl2379033057d&dq=&hl=en&lr=&ie=UTF-8&safe=off&selm=3A35424A.CA80743B%40univ.ox.ac.uk`. Current Feb. 2003.

[7] S. Mazzocchi. (Apache.org) Wiki RSS. `http://nagoya.apache.org/eyebrowse/ReadMsg?listName=community@apache.org&msgNo=946`. Current Feb. 2003.

[8] D. F. Sutherland, A. Greenhouse, and W. L. Scherlis. The code of many colors: Relating threads to code and shared state. In *PASTE'02*, pages 77–83, New York, Nov. 2002. ACM Press.

[9] D.-W. van Gulik. (Apache.org) Wiki - we have a problem :). `http://nagoya.apache.org/eyebrowse/ReadMsg?listName=community@apache.org&msgNo=1315`. Current Feb. 2003.