

# Coherent Software Configuration Management within a Project-Based Paradigm

Tim Howe  
Computer Sciences Corporation  
`thowe3@csc.com`

2 November 2007

## **Abstract**

Software Configuration Management is a requirement for any non-trivial software work. However, commonly used SCM tools tend to assume a workflow contradictory to the process of discrete service requests we use in our work. We integrate off-the-shelf tools within our own application to provide a development environment which tracks related work within and between SRs, and enable simultaneous compliance with various processes within CSC and on the client side. Our work provides insights into competing models of multi-project management and our tools allow clean and auditable resolution of individual conflicts.

This paper describes the specific needs and constraints we encounter in our team's work. It details the layout of our Subversion repository and the development of TaskMaster, the application we developed to interact with that repository. Technical details of the work are explored as well as future possibilities.

# Contents

<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>4</b>
<b>2 Software configuration management</b>	<b>4</b>
2.1 What is software configuration management? . . . . .	4
2.2 Use cases . . . . .	5
2.3 Version control models . . . . .	6
2.4 VC and SCM tools . . . . .	6
<b>3 Version control tool requirements</b>	<b>6</b>
3.1 CMM requirements . . . . .	6
3.2 Sarbanes-Oxley requirements . . . . .	7
3.3 Account-specific requirements . . . . .	7
3.4 Network requirements . . . . .	7
<b>4 Development history</b>	<b>8</b>
4.1 Unversioned collection . . . . .	8
4.2 Subversion deployment . . . . .	8
4.3 Scripted updates . . . . .	9
4.4 TaskMaster . . . . .	11
<b>5 TaskMaster</b>	<b>12</b>
5.1 Technical overview . . . . .	12
5.2 Features and benefits . . . . .	12
5.2.1 Uses Subversion as backing store . . . . .	12
5.2.2 Decoupled from backing store . . . . .	12
5.2.3 Tracks work by task/project ID . . . . .	12
5.2.4 Local work meta-repository . . . . .	14
5.2.5 Java GUI . . . . .	14
5.2.6 Web access . . . . .	14
5.2.7 Archive exporting . . . . .	14
5.3 The future . . . . .	15
5.3.1 One-stop access to lower-level VCS tasks . . . . .	15
5.3.2 Enhanced indexing . . . . .	15
5.3.3 More user interfaces . . . . .	15
5.3.4 Release management . . . . .	15
5.3.5 Wider development participation . . . . .	16
<b>6 Informing management</b>	<b>16</b>
6.1 Reporting . . . . .	16
6.2 Multi-project management . . . . .	16
6.3 Unambiguous status . . . . .	17

<b>7</b>	<b>Conclusions</b>	<b>17</b>
7.1	Methodology . . . . .	17
7.2	Tools . . . . .	17
7.3	Implications . . . . .	17
<b>8</b>	<b>Acknowledgements</b>	<b>19</b>
	<b>References</b>	<b>20</b>

## 1 Introduction

The work described in this paper was performed within the Client Computing GHPAT of the Sun Microsystems account. This team receives service requests (SRs) to customize and update a series of internal and third-party applications. Maintaining a repository of work is necessary to provide quality service but organizing this repository effectively presents a challenge.

The contractual process requires all work to be requested and specified by a discrete SR. Many of the documents involved are related only to the individual SR. As the work is completed, the SR is closed down and the source and object files are packaged for release and archival. In this change-oriented [8] model, it is sensible for work to be organized by SR.

However, the same applications and codebases are addressed repeatedly by successive SRs: vertical reuse [10, 9, 15]. With each SR, new versions of each codebase are released, and the codebase retains changes for the next release. This long transaction [8] model suggests organizing by codebase.

Finally, individual SRs often combine multiple codebases to provide a service, in a particular case of horizontal reuse [10, 9, 15].

The taxonomies of these respective models conflict. If organizing the team's work by SR (*Root - SR - codebase - release*) is selected, it becomes difficult to locate all the SRs affecting a particular codebase; conversely, if organizing work by codebase (*Root - codebase - SR - release*) is selected, collecting the work performed for a particular SR can be complex. This is "the tyranny of the dominant decomposition" [20].

This paper attempts to describe the specific factors affecting our team's software reuse and configuration management patterns. The growth and structure of TaskMaster, an internal tool to automate a resolution of competing organizational hierarchies, are discussed. Finally we examine future directions to be explored and the wider applicability of this tool; its development model and functionality.

## 2 Software configuration management

### 2.1 What is software configuration management?

Software configuration management is a superset of the concept of version control. Most readers are likely familiar with version control and its use in software development. Nevertheless a minimal treatment is in order as background.

It is not uncommon for a particular piece of software to have portions whose origins verge on "prehistoric" [22]; that is, code persists while its reason for being is unknown. This can lead to the failure of rewrite efforts which do not take real-world bug fixes into account [18]. Hence, having development history available reduces risk considerably.

Version control, in its most basic form, provides access to the change history of source code or other documents [25]. Most version control systems (VCSes)

also provide varying levels of access control and other features.

Software configuration management (SCM) “identifies, controls, audits, and reports modifications that invariably occur while software is being developed and after it has been released to a customer”, encompassing not only mere versioning of source code but “all information produced as part of the software process” [17]. This typically includes requirements documents, test plans and results, deployment details, and of course the source code.

## 2.2 Use cases

A key benefit of any SCM tool is the ability to track and control changes to any software configuration item (SCI) and to obtain historical versions of individual components, entire projects, or configurations [8, 17].

Particular real-life needs for these capabilities which have been required by this team include:

- The latest approved project description document (PDD) at the time of a change to source code needed to be obtained to document the reason for a particular change.
- Before release, the scope of all SCI edits performed in a project was audited to ensure adherence to the baseline.
- The customer requested a detailed list of changes provided in a specific product version.
- After identifying a bug, the customer wanted to determine when it was introduced, to discover what users might have begun relying on that behavior.
- A user reported problems with a product which did not appear in testing and could not be reproduced outside of the user’s environment.

The last example especially highlights the benefits of SCM. By obtaining configuration models, test scripts and results, variances of the user’s environment from test environments were identified and test scripts updated for future regression testing. The PDD of the project in which the work was performed was consulted to determine if support for the user’s environment had been required and therefore if the fix was new work or a correction of incomplete original work.

None of these situations would have been adequately addressed by reliance on mere source code versioning, a simple document repository, or even both. Integration of the two provided a consistent timeline and assurances that requirements changes resulted in complete and properly scoped implementation, and source code changes resulted in retesting, and finally that releases were discrete and unique.

## 2.3 Version control models

Models used by version control systems vary widely. They range from single-user single-system, through multi-user networked, to fully distributed. There are gradations in all of these; for example some multi-user systems follow a model where a single user “checks out” a file and it is locked until they “check in” that file (i.e., commit or abort their changes).

Even in systems which share the same interaction model, the semantics of the versioning system can vary. As an example, CVS gives each file in a repository its own revision number [3], while Subversion gives the repository as a whole a single revision number [7].

## 2.4 VC and SCM tools

There are a number of version control systems publicly available, each with different features [2].

CSC has some tools internally which perform aspects of SCM, such as Tracker, which stores current and historical status of a service request alongside artifacts such as project description documents (PDDs) and test scripts with their results.

# 3 Version control tool requirements

The initial need, and the core of the future work, was to set up a version control system to track source code and project documents. Given the process — described in Section 4.2 — by which our work began, only freely available tools were considered.

The primary candidates were CVS and Subversion, since the decentralized systems did not meet the primary goal of retaining a central repository of work [23, 6].

Other requirements arose from regulatory or contractual requirements, or the business needs of the account.

## 3.1 CMM requirements

CSC has an active requirement from Sun Microsystems to bring internal development processes up to level 3 (“Defined”) of the Capability and Maturity Model (CMM). Capability level 3 requires consistency *across* processes. The obvious prerequisite is capability level 2, which requires infrastructure and policies to ensure that practices and process improvements are institutionalized and retained [4].

Therefore any system used must provide for work done in one project to be carried into the future, such as in the case of a request to produce an updated version of a software package. Eventually, as processes are solidified, templates should be created to ensure that all projects and codebases are handled consistently.

### 3.2 Sarbanes-Oxley requirements

CSC and Sun Microsystems are both publicly traded companies in the United States and therefore subject to the Sarbanes-Oxley Act. While the Public Company Accounting Oversight Board (PCAOB) and Securities and Exchange Commission (SEC) have indicated that “IT controls should only be part of the SOX 404 assessment to the extent that specific financial risks are addressed” [24], the American Institute of Certified Public Accountants (AICPA) points out that “The use of IT also affects the fundamental manner in which transactions are initiated, authorized, recorded, processed, and reported.” [1].

Our team has little accounting involvement, but data regarding the speed, scope, and effectiveness of our work could easily be rolled up into higher-level reports and potentially into financial projections. Furthermore, we are responsible for packaging StarOffice, Sun’s office productivity software, the spreadsheet of which is surely used in the implementation of manual controls.

Control activities considered in an audit which could be applied to SCM include [1]:

**authorization** An auditor should be able to determine the client request which led to a specific change.

**segregation of duties** It should be possible to “hand off” the process at defined milestones, i.e., from development to packaging to testing to deployment.

**safeguarding** Integrity of the repository should be assured. Only authorized individuals should be able to make changes.

**asset accountability** An auditor should be able to determine when and by whom a change was made. The development process should ensure that best practices are followed in collaborative development.

### 3.3 Account-specific requirements

Sun Microsystems produces Solaris, a Unix operating system, and uses it extensively internally. Most work on the account is done on-site on Unix or Unix-like systems. Therefore any system used must work properly on Unix. Yet if any interaction with the repository is to be expected by personnel from CSC at large (e.g., project managers, quality leads) it must also be available and comfortable on Windows.

Sun uses CVS internally. Because of this and CVS’s wide historical use, many developers on our team are familiar with CVS commands. Therefore they would be at an advantage using CVS or a similar system.

### 3.4 Network requirements

Our team is globally distributed and includes developers with slow, expensive, or unreliable Internet or VPN access. Therefore the user must be able to achieve

a usable subset of behavior without an always-on connection to the repository, and network operations must be efficient.

## 4 Development history

It is important to note that development on what was to become TaskMaster began and progressed without an end goal or complete specification in mind. Rather, it grew organically, adding functionality as needed. Each step was an effort to refine current practice within the account and bring it in line with CSC's Global Project Framework (GPF), and to automate the resulting specific process.

A focus on incremental improvement ensures that at all times the tool retains its usefulness. By canonicalizing existing process, it allows the team to continue their natural working style [13] while ensuring and improving consistency [4].

### 4.1 Unversioned collection

Initially no version control was in use by our team. A Web server was available, and users could upload content by dropping it into the “document root” of the Web site. Unfortunately no versioning features were available, and weak NFS authentication coupled with shared root access meant that accountability for changes was missing as well.

Sun had its own CVS server, but it was a troublesome process to gain access. Organizing our work and keeping it from impacting theirs posed problems, as we were tasked with performing updates to software “owned” by other employee users. It also would not have been appropriate to upload internal CSC documents into the client system.

Finally we became aware of the desirability of maintaining a level of distance from the customer. In several cases, increased transparency into the development process led the client to become alarmed by viewing unfinished work, causing delays and general consternation. We also relied too much on that transparency and occasionally failed in communicating the availability of new releases to the client. Therefore it became necessary for us to seek a private repository, accessible on the client network, and put in place a defined release process.

### 4.2 Subversion deployment

This phase was simple: We allocated one of the servers set aside for exclusive CSC development use and installed Subversion, chosen for its explicit design goal as a “compelling replacement for [CVS]” [5] and the built-in Web interface.

The initial repository layout was simple. Subversion allows an arbitrary free-form layout; branches and tags are merely mental constructs in its model [7]. However we followed a fairly standard model with codebase rooted in `/codebase`, each containing `trunk`, `branches`, and `tags`.

```

/
codebase/
  nct/
    trunk/
      src/
      doc/
      package/
    branches/
      SR162210/
        src/
        doc/
        package/
    tags/
      1.1/
      1.2/
work/
  SR162210/
    project/
    src/
      nct -> /codebase/nct/branches/SR162210
      firefox -> ...
      ...

```

Figure 1: Original repository layout

However, that only addressed work on an individual codebase (or product), such as our packaging of the Firefox Web browser or our custom network configuration app for the Solaris x86 laptop client. Therefore we created a second root, `/work`, containing subdirectories indexed by SR number. Each project also had the standard directories within, to allow for milestones and the like. Project metadata could be stored here, with codebase branches linked in via Subversion “externals”.

The result is depicted in Figure 1.

### 4.3 Scripted updates

As the number of simultaneous active projects grew, we found ourselves switching from project to project, and having to perform numerous manual project checkouts. This grew tedious, especially when bringing in new developers with various levels of familiarity with version control systems.

The result was `get-active-SRs` (depicted in Figure 2), a simple shell script which checked out each project into a local work area. In this way all projects were easily accessible to every team member and kept up to date.

```

#!/usr/bin/env zsh

repository=http://csc-ccs-svn.sfbay.sun.com/svn

typeset -A active

for project in `svn ls ${repository}/work`
do
    project=`echo ${project} | cut -d/ -f1` # get rid of trailing '/'
    active[${project}]=                    # mark this project as active

    echo ${project}:

    if [[ -d ${project} ]]
    then
        svn update ${project}             # already checked out
    else
        svn co ${repository}/work/${project}/trunk ${project} # it's new; fetch it
    fi

    echo
done

for project in *
do
    if [[ -d ${project} ]]
    then
        if [[ -z ${active[(i)${project}]} ]]
        then
            echo ${project} is not an active SR
        fi
    fi
done

```

Figure 2: get-active-SRs

```

/
  codebase/
    nct/
      trunk/
        src/
        doc/
        package/
      branches/
      tags/
        1.1/
        1.2/
  work/
    SR162210/
      project/
      src/
        nct/
          src/
          doc/
          package/
        firefox/
        ...
        ...

```

Figure 3: Updated repository layout

#### 4.4 TaskMaster

Unfortunately this script had some limitations. It was difficult to run on Windows due to the simple command line interface and a requirement on Cygwin, a Unix emulation layer. The shell scripting language made it difficult to extend as well.

We then began development on TaskMaster, described in detail in Section 5. Originally the goal was to duplicate the functionality of the `get-active-SRs` shell script and add a graphical user interface (GUI). But a number of new features were added, and development continues as we meet different workflow challenges.

In the process we changed the repository layout from that mentioned in Section 4.2. While the use of externals allowed us to easily see which projects were touching a particular codebase, it made day-to-day work troublesome. For instance, one cannot commit a containing directory and have the externals committed as well; one must commit each external in turn. That alone made it impossible to make a coordinated change across several codebases and log the change against the project requirements.

We therefore took advantage of the freedom offered by Subversion and simply branched directly into the project directory, as shown in Figure 3.

## 5 TaskMaster

### 5.1 Technical overview

TaskMaster is a Java application providing a front-end to Subversion as well as some operations of its own. We use Java Web Start for deployment, allowing painless updates for everyone on our team.

All operations are abstracted away from the user interface and stubs currently exist for both CLI and GUI access. Currently, however, most operations are only implemented in the GUI instance.

Although Subversion provides Java language bindings, and SVNKit (formerly JavaSVN) is available as a native Java implementation, we call the `svn` binary in a subprocess. This is because the user already has to interact with `svn` for many operations, and so only has to set up connection parameters once. At the time we began development, JavaSVN provided a much lower-level API than it now does. Finally using the native client ensures immediate availability and compatibility of all Subversion features.

We do create a partial internal model of the repository and the working copies as needed, allowing an object-oriented approach to interacting with the repository. In fact, we could swap out Subversion for another VCS if needed. Our model represents not only Subversion concepts such as “repository” and “working copy”, but higher-level constructs such as “project” and “codebase” as well (shown in Figure 4).

### 5.2 Features and benefits

#### 5.2.1 Uses Subversion as backing store

Subversion is extremely reliable; it was designed as and is widely considered to be the successor to CVS [14, 12, 19, 21]. It offers a number of essential features that CVS doesn't, such as atomic commits spanning multiple files, binary deltas, rename tracking. At the same time, commands and most workflow use cases are retained from CVS, allowing users familiar with it (such as many ex-Sun workers on our team) to use it with little retraining.

#### 5.2.2 Decoupled from backing store

Any operations in the GUI could be ported to another version control system if needed. In this way the user experience can be retained in the face of changing technology.

#### 5.2.3 Tracks work by task/project ID

Codebases are checked out into particular tasks or projects. This allows multiple tasks that utilize the same code to progress without interference to each other. During closedown the code is merged back into the “trunk” allowing any future work to pick up from a consistent state.

```

class Project {
    /**
     * Branch a codebase's trunk into this project for work.
     *
     * @param repository      Repository being used
     * @param codebaseName    name of codebase to be branched
     */
    public void importCodebase(Repository repository, String codebaseName)
        throws RepositoryException
    {
        update();
        repository.branch(codebaseName, this);
        commit("branched codebase " + codebaseName + " for " + toString());
        update();
    }
}

class Repository {
    /**
     * Branch a codebase's trunk to a Project's codebase area.
     *
     * @param codebase        name of codebase to be branched
     * @param project         project to import the branch
     */
    public void branch(String codebase, Project project) throws RepositoryException {
        try {
            copy(new URL(Area.CODEBASE.getUrl(this), codebase + "/trunk/"),
                new File(Project.Subdir.CODEBASES.getFile(project), codebase + "/"));
        } catch (Exception e) {
            throw new RepositoryException("Error branching", e);
        }
    }
}

```

Figure 4: Workflow classes interacting

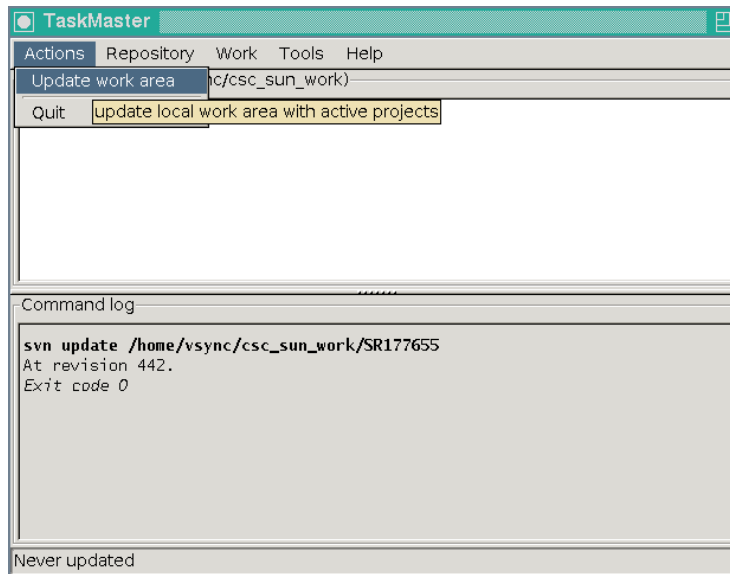


Figure 5: Updating the local work meta-repository

#### 5.2.4 Local work meta-repository

Everyone using the tool has a local repository with the current active work. This allows easy reference to the latest approved PDD and means anyone can lend a hand without a complicated checkout process. (Shown in Figure 5.)

#### 5.2.5 Java GUI

The GUI provides ease-of-use to team members in all roles, from management to technical. CSC project managers, business analysts, and quality leads can easily access the tool from their CSC-provided Windows laptops, while on-site engineers can use the same tool on Sun's laptops, workstations, or servers.

#### 5.2.6 Web access

The use of Subversion automatically provides Web-based access to the repository. This provides a significant benefit for users searching for comparisons to past releases, a repository overview, or for reusable software [16].

Further browseable functionality was achieved by integrating ViewSVN [11].

In-progress and future Web links are described in Section 5.3.2 and Section 5.3.3.

#### 5.2.7 Archive exporting

This simplifies the process of gathering documents and artifacts. Each codebase stores relevant test scripts and documentation in standard locations, and each

project does likewise for PDDs, estimates, and the like. With a simple mouse click the user can generate a consistent and up-to-date Zip archive for Tracker, or for the client's "SunWebCollab" system. CSC internal documents are not exported to the client.

## **5.3 The future**

### **5.3.1 One-stop access to lower-level VCS tasks**

At this point, TaskMaster only handles checking out or updating active work, as well as simple tagging and branching. For other tasks, such as merges and commits, the user must use the Subversion tools. On Windows TortoiseSVN provides an excellent integration with the Explorer shell. However it would be nice for basic tasks to be handled in the UI.

Upcoming versions will allow commits and similar functionality, as well as offer users the option of opening a graphical file browser or terminal on the selected task/project.

### **5.3.2 Enhanced indexing**

TaskMaster tracks each unit of work by its SR# or Bugster CR#. This is optimal as it provides a unique ID for indexing. However this is currently all that is presented in the UI. The result is that the user must maintain a mental list of SRs.

Upcoming versions will integrate with Dean Brundage's excellent customer-facing Web task list ("deantool") to obtain a short description which will be displayed to the user.

### **5.3.3 More user interfaces**

Currently the functionality offered by TaskMaster is only available through the Java application. Although portable, it requires a GUI and an installed Subversion client.

Currently a Zip format snapshot can be created as mentioned above. Stubs are already in place to implement a command-line interface (CLI). And upcoming versions will integrate with the "deantool" mentioned above to export commit messages to the client.

### **5.3.4 Release management**

While source code and documents are handled by Subversion, we have a separate artifact storage area to handle both vendor artifacts (e.g., binary applications we repackage, stock source code) and output artifacts (packages we deliver to the client). Work is underway to have TaskMaster automatically synchronize selected projects' artifact collections. As these files tend to be large, it is important that they only be delivered to users actively working on them.

An intriguing possibility is that of having a daemon process using the TaskMaster code managing build and test servers. The developer could then fire off “tinderbox” builds with a simple mouse click and deliver binaries to testing staff.

### 5.3.5 Wider development participation

Although design and code reviews were performed and continue with each new release, there are currently few developers doing significant development on TaskMaster. This presents vulnerability.

The code could probably be made more robust by opening the source either to a liaison engineer on the client side or in another GHPAT or account, or by open-sourcing the tool entirely (it has no CSC- nor Sun-specific code) with our team as a release gateway to perform QA and QC. If it is taken up by others in addition it will likely become more generalized and future-proof, and support will be more available for us and others.

## 6 Informing management

SCM generally and TaskMaster in particular offer a wealth of opportunities for management resources to improve the performance of our team as a whole.

### 6.1 Reporting

Developers can be required to update a project-level document and “check off” tasks as they are completed. Version control ensures that developers don’t trample each other, and the project manager can always obtain the most current status.

It is trivial to obtain a list of commits to any particular project or by a particular user, or to see when a given file or module was last changed. Questions as to responsibility for a piece of code are easily answered with the aptly-named “blame” subcommand of Subversion.

### 6.2 Multi-project management

Unfortunately at times conflicting SRs are being worked simultaneously, requiring concurrent work on the same codebase. While the SR-based decomposition of the repository properly requires each change to code be tied to a discrete client request, the team can also choose milestones or scheduled times at which to “sync up” the branches. The history of the merged code is tracked, and problem-causing merges can be reverted.

A risky portion of the SR may be handled in a branch, keeping for example a large rewrite from impacting delivery time on simpler subtasks.

The dispatcher can easily get a view of all open work and examine activity levels on each, shifting resources if necessary to ensure fair scheduling.

### 6.3 Unambiguous status

When a product is released it receives an immutable “tag” with its version number. This may be used to prevent ambiguity in bug reports and quickly determine what was last delivered to the client.

We have added a “merge to trunk” step to our closedown process in which we ensure development has concluded, then import the work that was done into the reference codebases touched by the SR, for future use. The project’s trunk is then removed, ensuring that no future work is charged to the SR. However the final state of the project is tagged and saved, offering a permanent reference.

## 7 Conclusions

### 7.1 Methodology

Software development requires appropriate tools, and what is “appropriate” may depend on the situation. The marketplace and open-source ecosystem offer a diverse selection, and off-the-shelf software can easily meet the core needs of our development processes.

Custom programming is required to fully match these general-use tools to our given situation, but that work can be done in phases and organically, responding to the needs of our team.

By taking advantage of available technology and our strengths in software engineering and integration, we have been able to provide repeatable results to our customer, effectively fulfill our internal requirements, and increase productivity.

### 7.2 Tools

Focusing on modularity within the TaskMaster codebase has allowed reorganization to implement revised semantics. Integrating with external tools provided functionality coverage for areas not fully developed in the front-end application and compatibility with the user’s external development environment.

The platform-independent approach to development by using Java for user interface and other functionality allows a diverse user base to collaborate effectively, even on unanticipated platforms. The use of HTTP as an underlying protocol has been a boon for users taking advantage of proxy servers or behind restrictive client firewalls.

### 7.3 Implications

Technology improves the management of development. However with different customers, internal and external, come a variety of processes and systems to satisfy. Just as software libraries and interfaces provide “glue code” between technology systems, so can software tools provide “glue” between processes, repositories, and artifact libraries. Selecting an appropriate central point of

control for project items allows consistent distribution of required artifacts to all destinations.

## 8 Acknowledgements

The author wishes to especially thank Tom Adam, for consistently pointing our work toward our higher-level goals and seeking process improvement; Linda Flores, for insisting on process and providing excellent information and a consistent corpus of work to institutionalize in code; Craig Johnson, for providing managerial support, encouragement, and wise advice on this project, both technical and personal; Dean Brundage, for his complementary work and contributions to this project; Frank Psotka, for providing testing and much-needed advice on our workflow as well as keeping this project focused on meeting needs of our active work; and everyone in the Sun Microsystems account Client Computing team, for their patience, bug reports, and excellent feature ideas.

Tom Adam, Craig Johnson, Diane Yee, Dean Brundage, Frank Psotka, Brendan Howes, Drue Pautz, and Stretch Adams all provided invaluable feedback during the editing process.

## References

- [1] AMERICAN INSTITUTE OF CERTIFIED PUBLIC ACCOUNTANTS. Statement on auditing standards No. 109: Understanding the entity and its environment and assessing the risks of material misstatement. In *AICPA Professional Standards*. AICPA, 30 June 2006, pp. 1611–1656. Available from: <http://www.aicpa.org/download/members/div/auditstd/SAS109.PDF> [cited 19 Oct. 2007].
- [2] BETTER SCM INITIATIVE. Version control system comparison [online]. 9 Mar. 2007. Available from: <http://better-scm.berlios.de/comparison/comparison.html> [cited 19 Oct. 2007].
- [3] CEDERQVIST, P., ET AL. *Version Management with CVS*, 1.11.22 ed., 2005.
- [4] CMMI PRODUCT TEAM. *CMMI for Development*, 1.2 ed. Carnegie Mellon Software Engineering Institute, Aug. 2006. Available from: <http://www.sei.cmu.edu/publications/documents/06.reports/06tr008.html> [cited 19 Oct. 2007].
- [5] COLLINS-SUSSMAN, B. The Subversion project: Buiding a better CVS. *Linux Journal* (1 Feb. 2002). Available from: <http://www.linuxjournal.com/article/4768> [cited 19 Oct. 2007].
- [6] COLLINS-SUSSMAN, B. The risks of distributed version control [online]. 10 Nov. 2005. Available from: <http://blog.red-bean.com/sussman/?p=20> [cited 19 Oct. 2007].
- [7] COLLINS-SUSSMAN, B., FITZPATRICK, B. W., AND PILATO, C. M. *Version Control with Subversion*, 1.4 ed., 2007.
- [8] FEILER, P. H. Configuration management models in commercial environments. Tech. Rep. CMU/SEI-91-TR-7, Carnegie Mellon Software Engineering Institute, Mar. 1991.
- [9] GISI, M. A., AND SACCHI, C. A positive experience with software reuse supported by a software bus framework. In *Proceedings: Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability* (24–26 Mar. 1993), IEEE Computer Society Press, pp. 196–203.
- [10] GOTZHEIN, R. Vertical reuse in the development of distributed systems with FDTs. In *Formal Techniques for Networked and Distributed Systems – FORTE 2003* (29 Sept. – 2 Oct. 2003), International Federation for Information Processing, Springer-Verlag, pp. 31–47.
- [11] HOKKANEN, H. ViewSVN - web interface for viewing subversion repositories [online]. Available from: <http://viewsvn.berlios.de/> [cited 1 Nov. 2007].

- [12] NEARY, D. Subversion: Building a better CVS. *Linux Magazine* (May 2003), 59–63.
- [13] NIZAMI, K. Global software development and delivery. *Dr. Dobb's Journal* (Aug. 2007), 22–28.
- [14] O'SULLIVAN, B. *Distributed revision control with Mercurial*. Mercurial project, 10 Sept. 2007, ch. 1.2 A short history of revision control. Rev. 8627f718517a.
- [15] POULIN, J. S. Software reuse on the Army SBIS program. *Crosstalk: The Journal of Defense Software Engineering* (July 1995), 19–24.
- [16] POULIN, J. S., AND WERKMAN, K. W. Software reuse libraries with Mosaic. In *2nd International World-Wide Web Conference: Mosaic and the Web* (17–20 Oct. 1994).
- [17] PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*, fourth ed. The McGraw-Hill Companies, Inc., 1997.
- [18] SPOLSKY, J. Things you should never do, Part I [online]. 6 Apr. 2006. Available from: <http://www.joelonsoftware.com/articles/fog0000000069.html> [cited 19 Oct. 2007].
- [19] SUBVERSION PROJECT. Subversion testimonials [online]. Available from: <http://subversion.tigris.org/testimonials.html> [cited 1 Nov. 2007].
- [20] TARR, P. L., OSSHER, H., HARRISON, W. H., AND SUTTON, JR., S. M.  $N$  degrees of separation: Multi-dimensional separation of concerns. In *International Conference on Software Engineering* (16–22 Mar. 1999), pp. 107–119.
- [21] TATHAM, S. My experiences with Subversion [online]. Available from: <http://www.chiark.greenend.org.uk/~sgtatham/svn.html> [cited 1 Nov. 2007].
- [22] WEINBERG, G. M. *The Psychology of Computer Programming*. Van Nostrand Reinhold Company, 1971.
- [23] WHEELER, D. A. Comments on open source software / free software (OSS/FS) software configuration management (SCM) systems [online]. 18 May 2005. Available from: <http://www.dwheeler.com/essays/scm.html> [cited 19 Oct. 2007].
- [24] WIKIPEDIA. Information technology controls, 2007. [Online; accessed 19-October-2007]. Available from: [http://en.wikipedia.org/w/index.php?title=Information\\_technology\\_controls&oldid=155595485](http://en.wikipedia.org/w/index.php?title=Information_technology_controls&oldid=155595485).
- [25] WIKIPEDIA. Revision control, 2007. [Online; accessed 19-October-2007]. Available from: [http://en.wikipedia.org/w/index.php?title=Revision\\_control&oldid=165357002](http://en.wikipedia.org/w/index.php?title=Revision_control&oldid=165357002).