Scmbug: Policy-based Integration of Software Configuration Management with Bug-tracking

Kristis Makris
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ, 85287
kristis.makris@asu.edu

Kyung Dong Ryu
IBM T.J. Watson Research Center
1101 Kitchawan Rd
Yorktown Heights, NY 10598
kryu@us.ibm.com

Abstract

Software configuration management(SCM) and bug-tracking are key components of a successful software engineering project. Existing systems integrating the two have failed to meet the needs of the ASU scalable computing lab, powered by open-source software. An improved solution to the integration problem, designed to accomodate both free and commercial systems alike, is presented.

Scmbug offers a policy-based mechanism of capturing and handling integration of SCM events, such as committing software changesets and labeling software releases, with a bug-tracking system. Synchronous verification checks and the flexibilty to match multiple development models separate this approach from related work. We address design limitations of existing integration efforts, suggest improvements in SCM and bug-tracking systems required to achieve a scalable solution, and document our early integration experiences.

1 Introduction

SCM is key[17] in maintaining quality in software engineering. SCM systems, or even simple version control systems, guarantee that a record of all changes and enhancements to software is maintained. When bugs creep in software, a trace of how changes occurred is available, simplifying the process of identifying regression points and correcting defects.

Paired with SCM, bug-tracking is another key in engineering quality software. Defect-tracking systems guarantee that nothing gets swept under the carpet; they provide a method of creating, storing, arranging and processing defect reports and enhancement requests. Those who do not use a bug-tracking system tend to rely on shared lists, email, spreadsheets and/or Post-It notes to monitor the status of defects. This procedure is usually error-prone and tends to cause those bugs judged least significant by developers to be dropped or ignored[10].

By examining a log of software changes from an SCM tool, it is uncertain *why* the changes occurred. By examining a defect report, it is uncertain *what* changed in software in response to the defect. Integration of SCM with bug-tracking ties the reason *why* a feature/defect was developed/fixed with *what* software changes occurred in the SCM system to accomplish this. Marrying the SCM and bug-tracking systems improves the traceability of software changesets, quality of documentation in defect reports, and quality of release documents.

SCM[11, 9, 18, 6, 5, 7, 3] and defect-tracking[2, 4, 1] systems are widely popular and in deployment. Commercial systems implement a mostly inflexible integration of the two, and free systems lack a variety of important verification guarantees. None of the existing systems met the integration demands of the ASU scalable computing lab. At a minimum, an integration system should provide:

- Integration of *common denominator* SCM events, such as committing changesets and labeling releases. When a changeset is committed, the accompanying log message should be inserted in the bug-tracker. The list of affected files, and the older/newer version numbers of each file, should be reflected in the bug report. When a release is labeled, the release version should be inserted in the bug-tracker.
- Synchronous verification checks of SCM

actions against the bug-tracker. If a developer attempts to commit a changeset against an invalid product or bug id, the commit activity should fail, and the developer should be informed immediately.
- Policy-based configuration. The integration system should be able to match the development model followed by an organization by tuning run-time parameters.
- Secure deployment of the integration over the public Internet.
- An interface to integrate any SCM system with any bug-tracking system, overcoming limitations of the involved systems where possible.
- A mechanism to produce a Version Description Document (VDD) detailing the defects corrected, and the changesets involved in fixing a defect, between two releases of a software.

Scmbug is designed to meet these requirements. It is implemented in Perl, an excellent, cross-platform, glue programming language, for UNIX-like systems, such as Linux, AIX, and Solaris. It currently supports integration of CVS and Subversion with Bugzilla.

The rest of this paper is structured as follows. Section 2 outlines the limitations of existing systems that prompted our solution. The system design is presented in Section 3, and a short example in Section 4. The system's components are analyzed in Sections 5, 6, and 7. Section 8 brings to light the improved quality of release documentation enabled by this system. Early experiences using Scmbug are documented in Section 9, and insight to future work is given in Section 10. Finally, Section 11 concludes this paper.

2 Related Work

In the realm of commercial products, Perforce[7] and ClearCase/ClearQuest[3] only offer integration of their own SCM and bug-tracking systems, in a proprietary way. They do not readily support integration with free SCM systems, such as CVS[11], Subversion[9], and Arch[18] or free bug-tracking systems such as Bugzilla[2] MantisBT[4], and AntHill[1]. Work enabling integration of Perforce with Bugzilla 2.0-16 is available[8], but the integration API lacks an abstract bug-tracker interface, and requires modifications released by the Perforce integration team every time the Bugzilla database schema changes. Additionally, integration with other bug-tracking systems requires implementing a communication interface with the target bug-tracker. Finally, the integration is unsuitable for deployment across the public Internet, for reasons explained in Section 5.1.

In the free software arena, the most frequently attempted integration involves CVS[11], the dominant open-source network transparent version control system, with Bugzilla, a free defect tracking software that tracks millions of bugs and issues for hundreds of organizations around the world. Steve McIntyre's website[21] documents an approach using trigger scripts to email CVS's actions to an account (Bugzilla Email Gateway) configured to parse the email and process it accordingly. This approach is not synchronous. If a user accidentally commits against the wrong bug number, or a bug against which he is not the owner, the SCM system will proceed with the commit action regardless. The user will not be given the option to correct her actions. Additionally, if the email gateway is not active, the developer will not be immediately aware that integration failed.

The somewhat dated CVSZilla[15] project also integrates SCM events produced by CVS with Bugzilla 2.10. It does not support integration of events produced by any SCM system in a generic way. Moreover, it modifies the Bugzilla schema, and as a result does not work with current versions of Bugzilla, such as 2.18. Finally, this integration is unsuitable for deployment across the public Internet.

John C. Quillan was first to conceive and implement synchronous verification checks and a VDD generator, in work that was never publicly released. He integrated CVS with Bugzilla, but his design did not support any SCM system with any bug-tracking system, or deployment over the public Internet. Scmbug incorporates these features, and proposes a more flexible design that can accommodate both free and commercial SCM and bug-tracking systems.

3 System Architecture

Scmbug is a client/server system. As shown in Figure 1, it consists of a set of SCM system hooks that capture standard SCM events and a generic glue mechanism of handling these events on the machine hosting an SCM repository. These events are translated into integration requests and transported to a server daemon. The daemon em-

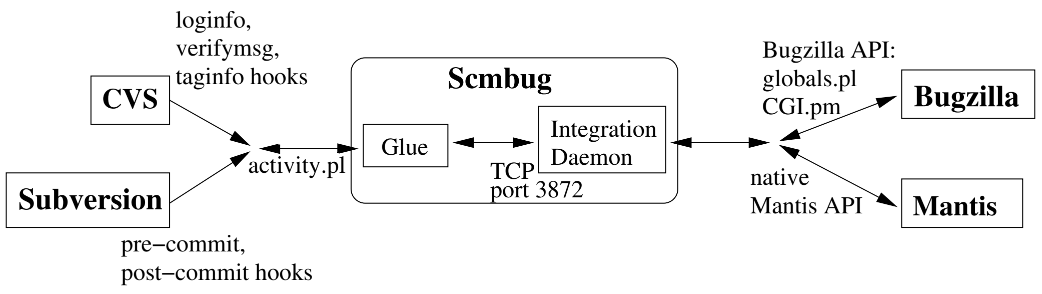

Figure 1: Scmbug system architecture diagram

ploys a generic mechanism of handling these requests and contains functionality that can process these requests per bug-tracking system.

The integration glue follows a common interface of handling SCM events, and groups the logic driving the behavior of the integration. SCM-specific implementation idiosynchracies, such as parameter decoding from SCM hooks, are isolated in separate modules. The abstraction of an integration glue is what permits any SCM system to be integrated using Scmbug. The integration functionality is always executed on the machine hosting the SCM repository.

Respectively, the integration daemon implements a common interface of accepting and processing integration requests. Bugtracker-specific functionality, such as reading metadata and updating bug comments, is implemented in separate modules. The abstraction of an integration daemon is what permits any bug-tracking system to be integrated using Scmbug. The SCM repository and the integration daemon can be hosted on separate machines.

4 Integration example

Lets examine an actual integration sequence using CVS and Bugzilla. Σ-Watch is a performance monitoring tool created by the ASU scalable computing lab. Monitoring the performance of a Zaurus SL-6000 PDA using this tool fails, and this defect is recorded in bug 417. After some initial investigation, the failure point is identified and documented in comment 6. A developer implements a fix, and attempts to commit his changeset using the command `'cvs commit'`. This command brings up a predefined log message template, in which the developer explains the source code change and enters the matching bug id (417) the changeset applies to, as shown in Figure 2.

Figure 3 presents the log comment stored in CVS, produced in a more readable Changelog format using the `cvs2cl` tool, in response to this changeset. This entry clearly details *what* changed: a global variable is introduced to hold the name of the network device. However, it does not explain *why* this change was required. *Why* must a global variable be used instead of a hardcoded value? To identify the reason *why* these files were modified one must consult the bug-tracking system. This log entry refers the reader to bug id 417.

Figure 4 captures the comment entered in Bugzilla in response to the changeset. The original SCM log message is included in the bug comments. The integration also generates a list of affected files, matched with the older/newer version numbers of each file, and appends it to the log message to show *what* changed in response to the bug fix. To identify *why* this changeset occured, one only needs to scroll in past history in Bugzilla to comment 6, with minimal effort, which reveals the root of the problem, as shown in Figure 5: PDAs use wireless, rather than ethernet, for network connectivity. The traceability of software changesets back to the root of problems is dramatically improved.

Note that a complete fix to the bug does not end with a single changeset. One day later, the developer drafts documentation on how the codebase can be configured with different network device names. He commits this new document in later changesets, in comments 13-16. By consulting this bug, the entire list of *what* changesets occured to handle the bug is readily available.

5 Integration Daemon

Inadequacies of existing systems called for a solution based on the abstraction of an integration daemon, as discussed next.

```
SCMBUG ID:   417
SCMBUG NOTE: Now using the name of the appropriate network device on
each system SigmaWatch runs. This name is set through a global
variable.
```

Figure 2: Documenting a fix for bug 417 during a `'cvs commit'`.

```
2004-08-21 Saturday 11:47  mkgnu

* src/host_node/userspace/server/readnet.c (1.9,
p_kpm_prior_to_bug424_fixes),
src/host_node/userspace/server/readnet.h.in (1.1,
p_kpm_prior_to_bug424_fixes), configure.in (1.33): SCMBUG ID:
417 SCMBUG NOTE:Now using the name of the appropriate network
device on each system SigmaWatch runs. This name is set
through a global variable.
```

Figure 3: Log comment entered in CVS for Bug 417, presented in a more readable Changelog format using `cvs2cl`. Through integration, this CVS entry is also documented in Bug 417, comment 10 of product Σ-Watch in Bugzilla, shown in Figure 4.

5.1 Public Internet Deployment

First, deployment over the public Internet was required to meet our development model. Members of our research group maintain individual SCM repositories on their personal laptops, to allow offline development. For our defect-tracking needs, a publicly available bug-tracking system is shared. Even though integration with bug-tracking is not possible when a user is working offline, it is still possible to produce the differences between a repository and a working copy (e.g. using `'cvs diff'`), and commit changesets at a later time. Hence, this method of development is encouraged. When problems arise and development of personal research prototypes stalls, pointing each other to a bug-report, for help, is beneficial. By describing a problem in a public defect-tracker, rather than a defect-tracker running on personal laptops, easy accessibility and quick resolution of defects are facilitated.

We also collaborate with researchers from different labs and research organizations on jointly funded projects, and do not wish to permit them access in our LAN using a VPN. Similarly, the security policy of collaborators sometimes does not grant us access to their internal resources. Using a public defect-tracker overcomes this problem, and reduces the administrative overhead of managing separate, private defect-trackers per project or organization. Experience has shown that limited labor and system administration expertise resources exist in an academic environment.

5.1.1 Minimizing exposure

Bugzilla, our defect-tracker of choice, uses MySQL as a database backend. Inserting log messages and version numbers in Bugzilla requires accessing the database. Opening the MySQL TCP port over the public Internet is a serious security risk, since it exposes access to other applications running on the database system. By using a separate integration daemon, and keeping this TCP port closed, only an integration interface is exposed.

5.1.2 Stand-alone backends

Some bug-tracking systems may not use a database backend that listens to a TCP port at all, such as the Berkley DB backend. Another example is debbugs[16], the bug-tracking software of the popular Debian Linux distribution. It uses it's own file-based database and is accessible through an email gateway. The abstraction of an integration daemon permits access to such file-based, stand-alone backends from the Internet.

5.1.3 Integration Security

Our development model does not match the vision of existing integration systems, such as Perforce, ClearCase, CVSZilla, or McIntyre's and Quillan's integration work. These solutions are deployed in a local area network environment and directly connect to the bug-tracking database. The general assumption is that no malicious users exist in the local network. Since the integration daemon listens on a public port, it cannot be assumed that all integration requests will originate

```
------- Additional Comment #10 From Kristis Makris  2004-08-21 11:47 -------

Now using the name of the appropriate network device on each system
SigmaWatch runs. This name is set through a global variable.



Affected files:
---------------
1.8 --> 1.9 system/src/host_node/userspace/server/readnet.c
NONE --> 1.1 system/src/host_node/userspace/server/readnet.h.in
```

Figure 4: Bug 417, comment 10 of product Σ-Watch in Bugzilla. The reason *why* this changeset occured was documented 2 days earlier in comment 6, shown in Figure 5.

```
 ------- Additional Comment #6 From Kristis Makris  2004-08-19 12:16 -------

The module that collects network information consults the hardcoded
"eth0" network device, which does not exist in PDAs. A "wlan0" should
be used in them instead, since they have wireless access; not
ethernet.
```

Figure 5: Root of the problem documented in bug 417, comment 6 of product Σ-Watch in Bugzilla. The problem is not documented in CVS.

from trusted developers of our lab.

The integration communication protocol includes the SCM username of the developer committing a changeset. This information is needed to identify the developer originating the software change. It is also needed to map to the username of the developer in the bug-tracking system. However, it is possible for an attacker to produce phony integration requests with properly matched usernames, and either pollute the bug-tracking system with misleading comments or add/delete product version numbers. As a remedy, public key authentication can be used to confirm the identity of integration requests in communications between the glue and the daemon.

5.2 Public Integration Interface

There's no public interface that system integrators can use to communicate with a bug-tracking system. Integration efforts currently duplicate functionality implemented in the codebase of the bug-tracking system. CVSZilla and Quillan's integration work followed this approach. When the database schema is modified in the next release of the bug-tracking system, the integration breaks. All SCM repositories must have their glue updated for the new bug-tracking system in order to proceed.

Instead, functionality from the bug-tracking system's codebase is directly reused. For example, a variable in the integration daemon configuration file points to the path were the Perl libraries distributed with the Bugzilla codebase are deployed. The Bugzilla-specific module of the integration daemon issues calls to this library. When the Bugzilla instance is upgraded, the new Bugzilla codebase includes functionality already updated by the bug-tracker's developers to match it's database schema. One only needs to point this path variable to the location of the new source distribution and restart the integration daemon. Therefore, an upgrade of the bug-tracking system does not require upgrading the glue in each SCM repository using this integration. Additionally, if a user migrates from Bugzilla to a different defect-tracking system she only has to specify the name of the new system in the daemon configuration file, and restart the daemon.

The bug-tracking system's codebase does not always provide all the functionality needed by the verification logic. For example, some queries specific to Bugzilla's database schema had to be implemented. Informing bug-tracker developers of such integration-related queries is important in providing generic verification logic. The Bugzilla developers are now planning[19] to use XML-RPC to define a public, HTTP-based interface for integration with 3rd party tools, which will include all the functionality needed by Scmbug.

Implementation of a similar interface is also missing in other bug-tracking systems, such as MantisBT and AntHill. Still, integration with other bug-tracking systems is desirable. Until other systems implement a similar interface, the daemon abstraction serves as a mediator, integration interface, where custom queries can be implemented.

5.3 Lack of SCM Integration Support in Bugtrackers

Free bug-tracking systems, such as Bugzilla, still lack support for SCM integration in their database schema. For example, integration work must be able to match the username used in the SCM system with the username used in the bug-tracking system to enforce a valid bug owner check, as described in Section 7.4. A bug-tracking system should provide space in it's database for each user's SCM username. Since this support is missing in some defect-trackers, the need for username mapping is satisfied by the integration daemon.

5.4 SCM System Limitations

The abstraction of an integration daemon also makes it possible to develop solutions to some known integration problems of existing SCM systems.

For example, CVS lacks atomic transactions. As a side-effect, when the same log message is used to commit files in two separate directories, two integration actions occur using the same log message. Duplicate log messages are then entered in the bug-tracking system. Coming back to the integration example of Section 4, Figure 3 shows that revision 1.33 of the file `system/configure.in` was produced during the example changeset. However, Figure 4 is missing a corresponding entry for this modification in the affected files list. This change was documented in comment 11 with a duplicate log message.

It is possible to solve this problem by caching integration requests at the daemon for a configurable, short time interval. Reception of another integration request with the same log message indicates that the transaction should have been atomic. The affected files list can be merged, and the log message inserted in the bug tracking system only once.

6 SCM Hooks

In modern version control systems, certain SCM events can trigger scripts that perform additional work handling the event. As a *common denominator*, SCM systems are expected today to offer hooks on the following events:

- `pre-commit`. Used to verify if the commit should proceed.
- `post-commit`. Used to integrate a commit log message with a bug-tracking system. Invoked only if `pre-commit` succeeded.
- `pre-label`. Used to verify if a creation of a tag or branch should proceed.
- `post-label`. Used to integrate the tag or branch name with a bug-tracking system. Invoked only if `pre-label` succeeded.

Scmbug installs scripts in an SCM repository which are executed as hooks for each event. After information about each event is collected, integration proceeds through the SCM glue logic. Integration of SCM systems that do not provide these hooks is not possible.

Subversion does not provide the `pre-label` and `post-label` hooks. Tags and branches are created using the `'svn copy'` command in predefined directories named `/tags` and `/branches`[13]. They are later committed with a regular `'svn commit'`. Nevertheless, it is still possible to detect during a `pre-commit` event if a tag or branch action is underway. If the commit activity indicates that a new subdirectory is created under `/tags` or `/branches`, then the transaction corresponds to a labeling action.

7 Integration Policies

Central to the behavior of the common glue logic is the notion of integration policies. It is essential that the glue prevents developers from describing erroneous integration actions. At the same time, the capability to match multiple development models is equally important. A configuration file stored in the SCM system controls the overall behavior of the integration glue, as described in the next sections.

7.1 Enabled Integration

The integration can be disabled at any time, by changing the value of a flag in the glue configuration file. Hence, the SCM system can easily back-out from using this integration.

7.2 Presence of Distinct Bug IDs

SCM systems prompt a developer with a log message template prior to committing a changeset. In order to integrate the log message of a changeset

```
SCMBUG ID:
SCMBUG NOTE:
```

Figure 6: Scmbug log message template. Essential in determining the bug against which a changeset is commited.

with a bug id in the bug-tracking system, a bug id must be included in a parseable format in the log message. Thus, Scmbug expects the predefined template shown in Figure 6 to be filled-in during a commit action.

Given a log message in this format, the glue logic verifies that one or more bug ids were supplied. Accepting multiple bug ids is required when a changeset fixes a collection of defects.

Additionally, the glue verifies that the ids supplied are unique. A duplicate bug id is most likely an indication that a developer typed the wrong bug id.

7.3 Valid Log Message Size

Project managers and SCM repository administrators often have to convince lazy programmers that there is value in typing a detailed log message during commits, and in SCM in general. An optional policy that requires the log message to meet a configurable, minimum message size is used to address this problem. Messages with a size less than the minimum cause an SCM commit action to be rejected and force a programmer to recommit with increased log comment verbosity.

Simply printing a warning when a small log message is entered and accepting the commit is not enough. A small, incomplete, misleading, or practically useless log comment can cause great grief when a bug has creeped in the software. Such inadequate documentation on a changeset introducing the bug can send a developer in a laborious SCM history cross-examination and bug-hunting trip.

7.4 Valid Bug Owner

Formal defect-tracking processes often define a Change Control Board (CCB)[14] or project manager that dispositions bug ids to developers based on the components they touch. In such settings, developers are not expected to commit changesets that touch bugs assigned to other developers. A policy in Scmbug verifies that the user issuing an integration action is the owner of the bug id specified in the bug-tracking system. This check ensures developers don't step on each other's assigned work.

7.5 Open Bug State

Even more important than valid bug owner checks is the capability to verify that a changeset is committed against a bug id set in an *open* state. For example, committing against a bug id that has been already resolved, marked either `FIXED` or `INVALID` (these are some resolution states in Bugzilla), would be wrong. Such verification checks alarm the developer that somebody else worked on the defect at hand already. Another example would be committing against a bug marked as `NEW` or `UNCONFIRMED`, where the CCB has not yet assigned the bug id to a developer. Examples of valid, *open* states would be `ASSIGNED` and `REOPENED`. This verification check ensures that a formal bug dispositioning process is followed.

7.6 Valid Product Name

A product name, specified in the glue configuration file, is transmitted to the integration daemon during an integration action. If the supplied bug id is associated with the specified product name in the bug-tracking system, the integration action proceeds. This verification check often captures cases where a developer enters the wrong bug id in a log message in an organization actively developing multiple products at the same time, such as Mozilla.org, and the GNOME and KDE projects.

This check assumes that an SCM system is used to host a single product. In some development models this may not be true. For example, a contracting company may choose to maintain documentation of multiple contracts in the same SCM system, while assigning different product names for them in the bug-tracking system. This approach reduces the administrative overhead of setting multiple SCM repositories. The valid product name policy also permits specification of multiple product names, to match such an alternative development model.

7.7 Convention-based Labeling

As software evolves through experimental, stable, or fork stages, the codebase may be labeled accordingly using the SCM system. Common labeling needs are:

- `Releases`. The codebase is tagged with a name indicating that it has reached a stable state, justifying a release point.

```
names => [
# Convention for official releases.
# For example:
# SCMBUG_RELEASE_0-2-7
'^.+?_RELEASE_[0-9]+-[0-9]+-[0-9]+$',

# Convention for development builds.
# For example:
# SCMBUG_BUILD_28_added_policies
'^.+?_BUILD_[0-9]+_.+$',

# Convention for branches.
# For example:
# b_glue_side_policies
'^b_.+$',

# Convention for private developer
# tags. Uses the developer's initials
# (either 2 or 3). For example:
# p_kpm_pre_bug353_fixes
'^p_[a-zA-Z][a-zA-Z]?[a-zA-Z]_.+$'
]
```

Figure 7: Label name convention examples. A list of regular expressions defines acceptable label names for releases, developer builds, forks and private labels.

- `Developer builds`. The codebase reached a developer milestone.
- `Forks`. A stable release will have important bug fixes supported in a separate branch. Forks may also be used to to implement experimental features without disrupting mainline development.
- `Private labels`. A developer labels the codebase either prior to or after introducing changesets that may introduce significant regression. The significance of the label is meaningful only to the developer.

In support of each of these labeling categories, a policy is introduced that ensures the label name used matches a configurable format defined as a regular expression. Figure 7 shows examples of label naming conventions that match these categories.

After a label name passes the convention check, it is entered in the bug-tracking system as an available version number on the specified product. The naming policy ensures developers apply a uniform labeling scheme. It also permits 3rd party tools to parse the available versions of a product in a consistent manner.

The convention-based labeling policy can be complemented by a role-based policy. A list of SCM usernames, in the form of a regular expression, authorized to create labels from each category could be specified. For example, only the release manager of a product should label releases and create forks, a group of high-ranked developers should label developer builds, and all developers should be encouraged to create private labels.

8 VDD Generator

Release management theory recommends software releases be paired with a document describing the changes since the previous release. Producing this document directly out of the SCM system, either using an automated tool, such as `cvs2cl` for CVS, or the SCM system itself, such as the `'svn log -r <rev_old>:<rev_new>'` command for Subversion, is not adequate. Changelog information derived strictly out of the SCM system is overly detailed. It describes software changesets at a lower, developer level, and is of little value to a user interested simply in a summary of added features. Moreover, when multiple changesets are committed in response to a defect such a Changelog document becomes lengthy. It takes considerable time to follow the history of changes and decipher if, or how, a defect was corrected.

It is more appropriate to pair such Changelog documents with a high-level summary of defects stored in the bug-tracking system. Summary fields are common in bugtrackers, but automating generation of this document involves determining exactly which bugs were worked on between releases. Identifying the date a release was labeled is a required first step (CVS does not support this), but it is not enough without integration of SCM with bug-tracking. Since the integration inserts log messages in the bug-tracking system, and the bug-tracker dates them, a list of bug ids that were worked on can be compiled simply by performing a date-range search on a product.

The bug-tracking system can then be requeried to report for each bug id not only the summary, but additional useful information. For example, it can report the resolution status(e.g. `FIXED`, `INVALID`), the bug owner(publicly attributing credit to the developer), resolution date, severity, priority, etc.

This report can also reflect decisions of the development team which are not documented in the SCM logs, such as choosing to not add a feature, resolving it as `WONTFIX`. It may also display bugs that were added in the period between releases but not worked yet, alerting users of newly discovered defects.

A tool that can produce such a version description document is under active development.

9 Early Experience

Scmbug has been deployed in our lab since March 2004 for integration of various research prototypes. In hindsight, we were late in developing a verbose logging mechanism in the integration daemon. Project deadlines and rapid development of Scmbug itself discouraged us from continuously upgrading to the latest version. We are unable to provide statistical information illustrating the frequency of developer integration errors that were caught by the verification policies.

9.1 Integration Upgrades

Seamlessly upgrading the integration work is not straight-forward. Between release 0.0.8 and 0.1.0 of Scmbug, the communication protocol between the glue and the daemon was altered. Upgrading the glue suddenly became a multi-stage process: (a) disable the glue, (b) install a newer glue release, (c) restart the integration daemon, (d) enable the glue. Disabling and enabling the glue was required to accept integration requests from a glueing codebase that matched the communication protocol understood by the integration daemon. It was also required for step (b) to succeed without communication with the daemon. Even though an automated glue installation and upgrading tool was employed, manual intervention was required on step (c) on the machine running the integration daemon. An important disadvantage is that upgrading multiple repositories requires all updated ones to remain without Scmbug integration until all repositories finish steps (a) and (b), before (c) is carried out. However, additional scripting by a system administrator could further automate this process.

After rapid development of our integration work, including rearranging of the Perl packaging structure installed in an SCM repository, the multi-stage process was proven to be defective in its implementation. Upgrading from release 0.1.1 to 0.3.1 revealed that the glue libraries used by the generic SCM hook processing script had changed package names and path, and the hooks were failing to carry out step (b). The solution to this problem was to completely remove the hooks rather than disable them through our configuration file.

Another design problem is that older glue installations are not preserved in the SCM repository. If a newer Scmbug release contains grave defects, one may not easily revert back to an older revision of the integration work. For example, CVS versions the hooks themselves. Committing a disabling hook, which is a hook that no longer invokes the glue processing script, still requires the original hook invoked for the last time. If this hook is defective, committing will fail. The defective hook will not be disabled, reaching a dead-end in upgrading the hook. In CVS one must then locally modify the repository to disable the hooks using separate RCS commands (RCS is the underlying database store of CVS). The installation mechanism can be enhanced to install permanently in an SCM repository every release of the integration glue, and allow switching between glue releases. This installation mechanism will need to be run locally on the machine hosting an SCM repository, to solve CVS's dead-end hooks problem. The integration daemon can also be enhanced to dynamically support older communication protocols.

9.2 Case Studies

Three undergraduate students in a microprocessor systems hardware course worked on one of our research prototypes for 16 weeks. They were introduced to our lab's development model, and a separate CVS branch was created for them to commit their changesets. The students worked on a total of 35 bugs. Being new to the concept of SCM, they occasionally checked out the main development line (often referred to as `HEAD`) instead of their personal branch. Consequently, they committed some changesets in `HEAD` instead of their personal branch. A policy for tuning fine-grained, branch-level, permission checks can solve this problem. For example, committing and labeling activities could be limited only to ourselves in `HEAD`, and students could be confined to committing only in their branch. Other developers using Scmbug also requested such a policy.

Development of another research prototype was integrated with Scmbug. Two developers experienced in the concept of SCM worked in the same lab, on 219 bugs, using CVS, for 15 weeks. Empirically, we can report that the most frequent verification check failed was a valid log message size check for 50 characters. This check correctly reminded the developers that they should be more verbose in documenting changeset history. The valid bug owner check was the second most frequent to fail. This was a result of miscommunication between developers. We speculate that this check may fail more often in open-source projects deploying Scmbug: the geographically distributed nature of developers in such a setting prohibits them from face-to-face communication on a daily basis. This check, along with the valid product name check, also failed due to typing errors while entering the bug id in the log message template. Finally, the least frequent verification check to fail was the open bug state check. This resulted from developer error in using TortoiseCVS[12], a GUI CVS client, to commit changesets. This client caches the most recent log messages. Developers often selected one of the previous messages in order to bring up the default log message template and type in a new message. After clearing out the old log message and typing a new one, developers occasionally forgot to change the bug id present in the cached log message. They instead attempted to commit against resolved bugs.

9.3 Just In Time Integration

Within five months of making Scmbug publicly available, we received over 6300 hits(49 per day) on the project's webpage[20]. The software distribution has been downloaded over 3200 times(26 per day). Other system integrators, including members of the Bugzilla integration team, discussed our design and recommended policy features which we implemented. Users reported deployment of Scmbug to successfully integrate both CVS and Subversion with Bugzilla, an activity that is largely simplified via an automatic installation script. It is clear that this integration tool fills a significant void in the open-source development community.

Why wasn't such an integration system built already? We were unable to find papers documenting the benefits of integrating SCM with bug-tracking, or proposing a similar solution. The Scmbug design is very flexible, and at the same time extremely simple. While the policy mechanism proposed is remarkably useful, it is not difficult to implement.

One reason might be that limited time is often available by developers for work on open-source projects. Sometimes, good-enough, quick hacks are easier to implement and preferred, rather than a full-blown, well-designed solution. For example, a common limitation of other systems integrating CVS with bug-tracking resulted from the inadequate mechanism CVS uses to provide the list of affected files in a commit trigger. A processing script using a single regular expression to parse these arguments gets confused if the filenames contain either commas or whitespaces. Scmbug handles this issue by employing a stateful parser, marginalizing the possibility of getting confused. The single regular expression method implemented in McIntyre's `bugzilla-watcher` script uses only 4 lines of Perl code for parsing filenames. Our parser was implemented in 145 lines of Perl code, and required significantly more time to develop.

A plethora of open-source SCM systems emerged in the past two years as alternatives to the currently dominant CVS. Subversion and OpenCM overcome various limitations of CVS and are still in active development. Arch was specifically designed for the distributed development needs of open source projects, such as the development model followed by the Linux kernel, and is still enhancing. Monotone is still in an alpha state and has been released only a year ago. We theorize that as these tools are still progressing to a stable state, integration with bug-tracking has yet to become a priority in their to-do list, and hence has not been addressed.

It is astonishing that mature, high-profile, public open-source software development websites still lack integration between their SCM and bug-tracking services. Some examples are Sourceforge (hosting over 93,100 projects, 983,900 users, in service for 5 years) and GNU Savannah (over 2,200 projects, 32,200 users, in service for 4 years). Our solution has been in high demand for a long time.

10 On-Going Work

OpenCM can commit changesets in disconnected mode[23], a feature also explored by Subversion developers. Laptop users can cache a repository and work offline. When connectivity is restored, they can synchronize their working set and resolve conflicts with the main repository. Scmbug could be improved to support a disconnected mode of integration. An integration daemon proxy, running on the user's laptop, could cache bug-tracking metadata required for policy verification checks, such as the list of bug ids, their state, bug owner, etc. All integration activity generated by disconnected commits could also be cached and synchronized later with the bug-tracker.

The star-topology development model assumed by Arch increases the significance of integrating branch and repository names. In this model, changesets may be produced by distributed SCM repositories, maintained by separate development teams. For example, a public Bugzilla instance tracks[22] defects in the Linux Kernel. Independent developers and companies maintaining private forks of the kernel post patches to bugs reported by anyone. However, they do not always report the name and branch of their repository fixing the bug. A user inspecting a bug report is uncertain which public tree includes the fix. Capturing a `<branch, repository name>` pair, which uniquely identifies the source of a changeset, could be supported by Scmbug. Providing generic support for this integration, would require all SCM systems to pass as arguments in integration hooks this information. The dominant CVS system, does not provide this information.

Improvements in SCM and bug-tracking systems are critical for a successful, generic integration solution. These are: (a) Section 5.2's public integration interface by bug-tracking systems, (b) the SCM username support of Section 5.3 by bug-tracking systems, (c) capturing the date a release was labeled by SCM systems, as mentioned in Section 8, (d) supplying a default log template in SCM systems, required by Section 7.2 (Subversion does not support this), the SCM hooks of Section 6 and (e) reporting in SCM hooks the `<branch, repository name>` pair just mentionied.

On-going work on Scmbug includes: (a) the public key authentication scheme of Section 5.1.3, (b) the role-based policy described in Section 7.7, (c) the fine-grained, branch-level permission policy mentioned in Section 9.2, (d) the VDD generator of Section 8, and (e) the improved upgrading mechanism of Section 9.1.

11 Conclusion

We presented Scmbug, a system offering policy-based integration of software configuration management with bug-tracking. Integration of SCM with bug-tracking improves the traceability of software changesets, the quality of documentation in defect reports, and quality of release documents.

Scmbug integrates activities such as committing software changesets and labeling software releases. The integration policies can be tuned to match multiple development models and provide synchronous verification checks. The design is flexible enough to support any SCM system with any bug-tracking system, can be deployed over the public Internet, improves the quality of release documentation, and can overcome limitations of existing systems. Finally, improvements in SCM and bug-tracking systems that are critical for a successful, generic integration solution are suggested.

12 Acknowledgments

Development of this system and paper benefited tremendously from insight on SCM and bug-tracking issues, along with invaluable feedback, from John C. Quillan, Mark S. Reibert, Dave Swegen, and the Bugzilla and Subversion developers. We regret Quillan's integration was never publicly released. We are convinced his work could have evolved to what Scmbug is today, and would have saved us considerable research effort.

We would also like to thank the existing Scmbug users for their increasing interest, testing, and continuous support of this integration system, especially Jay Glanville for his fruitful feedback.

13 Availability

Scmbug is free software, licensed under the GNU General Public License (GPL). It is available from `http://freshmeat.net/projects/scmbug`.

References

[1] AntHill. http://freshmeat.net/projects/anthill/, 2004.

[2] Bugzilla. http://www.bugzilla.org, 2004.

[3] ClearCase. http://www-306.ibm.com/software/awdtools/clearcase, 2004.

[4] MantisBT. http://freshmeat.net/projects/mantis/, 2004.

[5] Monotone. http://freshmeat.net/projects/monotone/, 2004.

[6] OpenCM. http://freshmeat.net/projects/opencm/, 2004.

[7] Perforce. http://freshmeat.net/projects/perforce/, 2004.

[8] Perforce Defect Tracking Integration. http://freshmeat.net/projects/p4dti/, 2004.

[9] Subversion. http://subversion.tigris.org, 2004.

[10] The Bugzilla Guide. http://www.bugzilla.org/docs/2.18/html/, 2004.

[11] The Concurrent Versions System (CVS). http://www.cvshome.org, 2004.

[12] TortoiseCVS. http://www.tortoisecvs.org, 2004.

[13] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly Media, 2004.

[14] Susan Dart. Concepts in configuration management systems. In *Proceedings of the 3rd international workshop on Software configuration management*, pages 1–18, 1999.

[15] Tony Garnock-Jones. CVSZilla. http://homepages.kcbbs.gen.nz/˜tonyg/, 2000.

[16] Ian Jackson. debbugs. http://www.chiark.greenend.org.uk/ ian/debbugs/, 1994.

[17] Gregor Joeris. Change management needs integrated process and configuration management. In M. Jazayeri and H. Schauer, editors, *Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97)*, pages 125–141. Springer–Verlag, 1997.

[18] Tom Lord. Arch revision control system. http://freshmeat.net/projects/archrevctl/, 2004.

[19] Kristis Makris. Provide an interface for SCM integration. Bugzilla Bugzilla, ID 255400, 2004.

[20] Kristis Makris. Scmbug. http://freshmeat.net/projects/scmbug/, 2004.

[21] Steve McIntyre. CVS/Bugzilla integration. http://www.einval.com/˜steve/software/cvs-bugzilla/, 2004.

[22] OSDL. Linux Kernel Bug Tracker. http://bugme.osdl.org/, 2004.

[23] Jonathan Shapiro and John Vanderburgh. CPCMS: A Configuration Management System Based on Cryptographic Names. In *2002 USENIX Annual Technical Conference, FREENIX Track*, 2002.