Edwards, N., Jongsuebchoke, D. and Storer, T. (2019) Sciit: Aligning Source Control Management and Issue Tracking Architectures. In: 35th IEEE International Conference on Software Maintenance and Evolution (ICSME 2019), Cleveland, OH, USA, 30 Sep - 04 Oct 2019, pp. 402-405. ISBN 9781728130958 (doi:10.1109/ICSME.2019.00069)

The material cannot be used for any other purpose without further permission of the publisher and is for private use only.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

http://eprints.gla.ac.uk/192034/

Deposited on 08 August 2019

Enlighten – Research publications by members of the University of Glasgow
http://eprints.gla.ac.uk

Sciit: Aligning Source Control Management and Issue Tracking Architectures

Nystrom Edwards, Dhitiwat Jongsuebchoke and Tim Storer
School of Computing Science
University of Glasgow
Glasgow, United Kingdom

***Abstract*—This paper presents sciit, a distributed issue tracker. Distributed issue tracking eliminates much of the friction that is otherwise necessitated by separately maintaining source code in a distributed source control management system (SCM) and task information in a centralised issue tracker. Sciit goes beyond the state of the art in distributed issue tracking by treating issues as first class change control items, represented as fragments of text anywhere within the SCM. This approach treats issues as representations of work in progress alongside other project artefacts. This alignment allows much of the meta-data about an issue, such as status, affected components and participants to be inferred directly from the state of the SCM, rather than requiring maintenance of this information by a developer. The paper presents a scenario to illustrate the benefits of sciit and an outline of the tool's architecture.**

***Index Terms*—distributed issue tracking; source control management; software life cycle**

I. Introduction

A centralised issue tracker has become the de facto means of coordinating work effort in agile software projects. In a typical workflow, project tasks are logged as *issues*, small self contained packages of work. These may then be further elaborated by the project team and annotated with additional meta-data, such as type (feature, bug etc.), priority, milestone, estimated effort and affected project components. Issues are then assigned to a developer for resolution. The developer works on the necessary changes, gradually making commits to a branch in the project's source configuration management system (SCM). Eventually, the work is completed and the developer requests for the changes to be reviewed within the team's quality assurance (QA) process. Once approved, the changes are merged from the feature branch into the main line of development, via a merge or pull request. Once these changes are accepted the issue status can be manually changed to resolved and the issue archived.

Reviewing this workflow reveals that software developers are required to manually maintain the consistency of information between their SCM and issue tracker. Curating this information accurately can substantially reduce the costs of, and increase the probability of issue resolution [3]. Unfortunately, this requires additional effort that may lead to information in different systems that is incorrect, incomplete, inconsistent or out of date [4].

For example, when work begins on an issue, a developer must create a new branch in the SCM and the issue must be marked as 'In progress' in the issue tracker. When an issue moves to QA, the developer must remember to mark it as 'in QA' in the issue tracker, then merge the work to a QA branch, or create a merge request. Finally, when an issue is resolved by merging a development branch into production, the developer must also remember to record this step in the issue tracker by changing the status to 'Resolved'.

Fundamentally, the redundancy arises because of the mismatch between the distributed architecture for source control management and the centralised architecture for issue tracking. We therefore argue that issues are early stage representations of software development work, that have strong dependencies on artefacts within the SCM and will eventually *be represented through alterations to artefacts within the SCM*. Issues should themselves therefore be treated as first class configuration items. All the changes necessary to realise the feature can be tracked in the SCM as change sets with associated commit messages. Storing the description of the work to be done as an early representation of the consequent implementation allows the associations between the two and their concurrently developing histories to be explicitly, rather than incidentally, linked. Many of the items of project meta-data, such as issue status can also be inferred from change sets without requiring data to be manually maintained. For example, once a feature has been fully implemented it can be simply deleted in the last change set that realises the change. As a further consequence, reliable issue tracking and associated work history record allows for more reliable project-scale metrics and better informed decision making.

Contribution: We have designed and implemented a proof of concept tool, sciit, based on the popular Git SCM. We describe the design of sciit and demonstrate how the features presented eliminates much of the friction necessary in issue-tracking based project coordination and enable many of the desirable items of information concerning project management to be inferred directly from the history of change sets within a project. Alongside the paper we provide a complete implementation of sciit as an open source project that combines a command line user interface extension to git and integration with the GitLab issue tracking web application to enable convenient migration[6].

This paper is structured as follows. Section II presents the workflow for interacting with sciit and illustrates how many aspects of redundancy in the software project management

```
@issue photo-upload-on-claim
@title Photo Upload on Claim
@description
 We need a photo upload feature so that
 customers can provide supplemental
 evidence for their insurance claim, e.g.
 of damage during a Road Traffic Accident.
@priority medium
```

backlog/upload-photo.md

(a)

```
+ git-sciit.exe issue photo-upload-on-claim

Title:             Photo Upload on Claim
ID:                photo-upload-on-claim
Status:            Open (Proposed)

Last Change:       twsswt | Wed Oct 31 16:21:55 2018 +0000
Created:           twsswt | Wed Oct 31 16:21:55 2018 +0000

Participants:      twsswt
Priority:          medium
Size:              335
Latest file path:  docs/issues/photo-upload-on-claim.md

Description:
 We need a photo upload
 feature so that customers can
 provide supplemental evidence
 for their insurance claim,
 e.g. of damage during a Road
 Traffic Accident.
*************************************************************
```

(b)

Fig. 1. Example of an issue in the backlog in sciit. The issue is created as a markdown formatted text file in a directory containing all the issues in the project backlog (1a). The issue as presented by the sciit command line tool combines information from the markdown file and the Git repository's meta-data (1b).

workflow can be eliminated by treating issues as source control items within the SCM. Section III presents the design and implementation of the sciit tool and its integration with the Git and GitLab packages. Finally, Section IV briefly describes the advance sciit makes compared with existing distributed issue trackers and outlines future work.

II. Example of using Sciit for Issue Tracking

We present an example scenario using sciit to illustrate the benefits of embedding issues in SCM artefacts. In the scenario, a product owner identifies new user stories concerning uploading photos for an insurance claims application. The product owner creates a new issue using the GitLab issue tracker interface. Sciit detects this and creates a new branch for the issue in the project repository on the server, creates a new markdown file in the `backlog` directory of the project repository source (as shown in Figure 1a), adds the file to the index and performs a commit.

Later, a software developer then pulls updates from the team's GitLab server to their local repository and receives the new issue. Figure 1b shows the new issue on the command line user interface. Notice that the summary presented by sciit contains extra information about the issue, such as the issue participants, that is not present in the text file itself, but is instead inferred from the version control system meta-data. The issue status is also reported as *Open (proposed)* because sciit interprets the presence of an issue in a feature branch, but not in master, as having not yet been accepted into the project. Whenever the branch is merged to master (e.g. through a pull request) the issue will be interpreted as *Open (accepted)*.

The developer decides that the acceptance test suite must be extended to include the new feature. A new issue is therefore created within a Gherkin feature file (Figure 2a), since sciit allows issues to be defined anywhere within a programming language specific comment format. The location of the issue is shown to the developer in the sciit interface, giving the user information about where the work will need to be done to resolve the issue. Sciit reports the issue *Open (in progress)* as soon as the developer makes a commit to the branch after the initial merge to master, as shown in Figure 2b. The branch can be pushed to the team's GitLab server so that other developers can receive the new issue.

The developer eventually completes all the work for the feature. At this stage the issue comment is simply deleted and a commit is made to the feature branch in the SCM. Reviewing the status of the project in sciit, the developer see that the issue's status is now reported as *Open (In review)*. Sciit assumes that since the issue no longer exists in the development branch, the author is awaiting approval for the completed changes to be merged into master. Finally, the feature passes the team's QA review process and the developer merges the branch into master, which causes the issue to be deleted from master. The issue is now marked as *Closed (resolved)*, as shown in Figure 2c. Alongside the change in status, sciit also automatically adds a closed time to the issue.

At the end of the sprint, the whole team reviews progress on the project. The full revision history of an issue can be reviewed using the command line interface or in the GitLab web application. For each change made to an issue, sciit reports the identifier of the contributor who made the commit, the date of the commit, an itemised list of changes to the issue and the commit message summary. The set of commits that the issue was present in is also listed. The issue tracker also reports the full history of all branches and file paths that the issue resided in during its life cycle. Supplementary metrics, such as the duration of the issue (the time between the first in progress commit and the issue being closed) are also reported.

III. Implementation

Figure 3 illustrates the software components of sciit, implemented on top of a peer-to-peer network of git repositories. The figure shows that sciit comprises a library for managing a cache of *issue snapshots* extracted from a git repository; a command line user interface git extension; and a GitLab integration that ensures consistency between a GitLab issue database and sciit git issue repository.

Sciit Library is a python package implemented on top of git-python [7]. The library has two main functions: maintaining

```
#***
# @issue photo-upload-on-claim-uat
# @title Photo Upload on Claim UAT
# @description
#  Extend existing claim user stories with
#  scenarios that include photo upload.
#***


Feature: Photo upload for claims

Scenario: Small JPEG Upload with Description
 Given a claim
 And a small JPEG
 And a description
 When I select the photograph
 And I enter a description
 And I click submit
 Then the photograph is stored
 And a database entry is created.
```

features/claim.feature

(a)

```
$ git sciit issue photo-upload-on-claim-uat

Title:              Photo Upload on Claim UAT
ID:                 photo-upload-on-claim-uat
Status:             Open (In Progress)

Last Change:        twsswt | Thu Dec 06 23:17:22 2018 +0000
Created:            twsswt | Thu Dec 06 22:58:39 2018 +0000

Participants:       twsswt
Size:               672
Latest file path:   features/claim.feature

Description:
  Extend existing claim user stories
   with scenarios that include photo
   upload.
***************************************************************
```

(b)

```
$ git sciit issue photo-upload-on-claim

Title:              Photo Upload on Claim
ID:                 photo-upload-on-claim
Status:             Closed (Resolved)

Closed:             twsswt | Thu Jan 17 18:52:19 2019 +0000
Last Change:        twsswt | Thu Jan 17 18:52:05 2019 +0000
Created:            twsswt | Thu Jan 17 18:51:34 2019 +0000

Participants:       twsswt
Priority:           medium
Blockers:           photo-upload-on-claim-uat(Closed), photo-
Size:               1612
Latest file path:   backlog/photo-upload-on-claim.md

Description:
 We need a photo upload
 feature so that customers can
 provide supplemental evidence
 for their insurance claim,
 e.g. of damage during a Road
 Traffic Accident.
***************************************************************
```

(c)

Fig. 2. Making progress on an issue. The developer extends the feature file in the sub-issue branch (2a). Sciit detects that a commit in the feature branch means that work on the issue is now ahead of the status in master and so reports the feature as *Open (In progress)* (2b). Sciit reports issues as Closed (resolved) when the issue is deleted from master (2c).

a cache of issue *snapshots* as commits are added to the repository; and reconstructing an issue history from the cache when interrogated by clients.

The cache update mechanism is invoked on a repository following commits, merges or pulls. Sciit is notified of these events by hooks in the host git repository. Sciit first determines which commits in the repository have not yet been cached. The set of changed file paths for each of these is then extracted from the repository. Each file is then scanned for issues marked by an `@issue` tag. Sciit assumes that issues are contained within comment strings of the language indicated by the file extension. Issue snapshots in files with a .java extension are assumed to be contained within `/**..**/` comments, for example. Each issue snapshot is then cached to an Sqlite database within the git repository.

Sciit uses the cache of snapshots taken from commits to build the history of issues. All snapshots are retrieved from the cache and grouped by issue id. The current status of each issue for many fields, such as title, description and last modification can be extracted directly from the latest snapshot for an issue. More complex properties can be inferred from analysis of the entire snapshot history. For example, the set of participants in an issue can be inferred from the usernames of commit authors who made modifications to issues. The status of an issue can be inferred by its historical presence in different branches in the project repository. The entire history of changes to an issue can also be extracted from the snapshots.

Git Extension is a git extension that enables users to review the status of issues in their repository. Issues can be presented in several different views including a status summary of all issues in the repository; a tracker view providing detailed information about a single issue, including its full history; and a log view providing a summary of any changes to an issue in a single commit.

The command line user interface can also be used to initialise and, if necessary, rebuild the sciit issue snapshot cache. Several commands have also been implemented to combine workflows steps that would otherwise have to be executed separately. For example, the `new` command creates a new branch, creates a new markdown file for an issue, performs a commit and then returns the user to the starting branch. This command also supports the creation of sub issues by enabling further issue branching from feature branches.

GitLab Integration allows GitLab to be used as a user interface for issues stored in the git repository, ensuring they are consistent with issues stored in the web application database. The integration updates the status of the GitLab issue database when commits are received on the server following a push from a client; and makes commits to the repository as a result of changes to the GitLab issue database. Interaction with GitLab is performed by a GitLab communicator module that exploits the GitLab REST API. This can be used to be notified of changes to information stored on GitLab and make changes to the issue database.

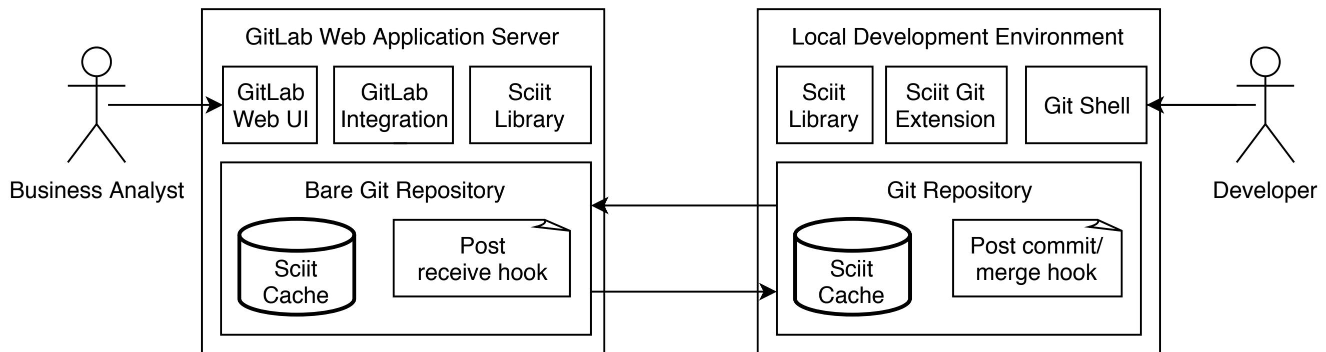

Fig. 3. The architecture of the sciit package within the Git/GitLab ecosystem.

IV. Conclusions

Issue trackers play a pivotal role in the coordination of a software project. Nonetheless, they are a significant source of redundant information because much of the information can already be inferred from the SCM, the primary representation of work for a software project. This paper presents sciit, a distributed issue tracker, in which issues are treated as first class control items within a SCM. Although a particular software workflow and issue type were used to illustrate sciit, the purpose is to demonstrate that the status of *any* software task and workflow is best elicited from the focus of that work, the source control repository, rather than secondary systems, such as issue trackers.

There are several existing tools that support distributed issue tracking, with a list maintained in the Sciit README file [6]. Some approaches, such as the Git-issue git extension [2], provide a similar command line interface to sciit, allowing users to manage issues within a git repository. Issues are stored in the git data structure and changes to issues can be migrated between repositories via push/pull operations. Others, such as the Bugs Everywhere project [1] provide issue tracking facilities on top of a number of different SCMs. A third category, including Fossil [5], are SCMs in their own right, with integrated issue tracking and wikis.

The architecture of sciit is an advance on these existing distributed issue trackers. First, sciit allows issues to be treated just the same as any other artefact in the source control system, rather than providing a separate data storage format for them. Issues can be created anywhere within a text file in the SCM repository, allowing a closer alignment between sciit issues and the work to be undertaken (or completed). Second, sciit leverages the close alignment of issue tracking content and other artefacts to allow many of the properties of an issue, such as status, component and participants to be inferred directly from the state of the issue in the repository, rather than requiring these to be updated manually. Unlike other approaches, Sciit relies on git for primary data storage, but also maintains a cached representation of issues drawn from git commits in order to present information efficiently.

We plan further integrations for sciit with popular software engineering tools, including IDEs and web based issue trackers to continue to develop sciit. Several new features have also been proposed, such as the inference of more types of meta-data based on the state of the SCM. Examples include exploiting change set characteristics (such as files and/or lines changed) to estimate costs for tickets.

We also plan a more extensive study of the benefits of using distributed issue tracking within real software projects. The dearth of academic literature in this area is a significant gap. Case study research of such experiences is necessary to understand how to best adapt existing software workflows to distributed issue tracking. Studies of this form also provide insights as to potential analytics for a project that can be derived from issue tracking data stored in sciit. We anticipate that longer term studies will generate significant insights for the development of successful distributed issue tracking systems.

References

[1] “Bugs everywhere project,” Available at http://www.bugseverywhere.org/, November 2012.

[2] “Git issue,” Available at https://github.com/dspinellis/git-issue, 2019.

[3] P. Hooimeijer and W. Weimer, “Modeling bug report quality,” in *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, R. E. K. Stirewalt, A. Egyed, and B. Fischer, Eds. ACM, 2007, pp. 34–43.

[4] M. Korkala and F. Maurer, “Waste identification as the means for improving communication in globally distributed agile software development,” *The Journal of Systems and Software*, vol. 95, pp. 122–140, 2014.

[5] J. Schimpf, *Fossil Version Control A Users Guide*, 2nd ed., Pandora Products, 215 Uschak Road, Derry, PA 15627, United States of America, November 2012, available at http://www.fossil-scm.org/schimpf-book/doc/2ndEdition/fossilbook.pdf.

[6] “Sciit,” Available at http://gitlab.com/sciit/. Accessed 22nd March 2019.

[7] M. Triers and S. Thiel, *GitPython Documentation*, 2015, available at https://gitpython.readthedocs.io/en/stable/.