Science of Computer Programming 96 (2014) 84–106

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Supporting streams of changes during branch integration

Verónica Uquillas Gómez [a,b], Stéphane Ducasse [a,b,*], Andy Kellens [a,b]

[a] Software Languages Lab, Vrije Universiteit Brussel, Belgium
[b] RMoD, Inria Lille – Nord Europe and Université de Lille, France

ARTICLE INFO

Article history:
Received 12 November 2012
Received in revised form 11 July 2014
Accepted 15 July 2014
Available online 30 August 2014

Keywords:
Branch
Source code changes
Stream of changes
Change dependencies
Merge

ABSTRACT

When developing large applications, integrators face the problem of integrating changes between branches or forks. While version control systems provide support for merging changes, this support is mostly text-based, and does not take the program entities into account. Furthermore, there exists no support for assessing which other changes a particular change depends on have to be integrated. Consequently, integrators are left to perform a manual and tedious comparison of the changes within the sequence of their branch and to successfully integrate them.

In this paper, we present an approach that analyzes changes within a sequence of changes (stream of changes): such analysis identifies and characterizes dependencies between the changes. The approach identifies changes as autonomous, only used by others, only using other changes, or both. Such a characterization aims at easing the integrator's work. In addition, the approach supports important queries that an integrator otherwise has to perform manually. We applied the approach to a stream of changes representing 5 years of development work on an open-source project and report our experiences.

1. Introduction

Version control systems (VCS) such as SVN, CVS and Git have become an indispensable tool for enabling teams of software developers to work together on a shared or distributed code base. Next to providing facilities for managing the source code of a system and maintaining that source code's history, these version control systems allow developers to work in *separate branches* of the system that later can be merged into the mainline of the system. Git, which is becoming increasingly popular, has placed branching at the center of its architecture and philosophy.

However, the task of understanding the consequence of a merge remains mostly manual and tedious due to a lack of practically applicable advanced tools. First, merging techniques used by popular VCS (*e.g.*, CVS, Subversion, Git) are based on simple, text-based algorithms, that only solve conflicts based on textual similarity, and are therefore *oblivious to the program entities* they merge. Even though there exist other approaches providing advanced merging support [2,17] that significantly reduce the amount of merging conflicts, such approaches do not support integrators in identifying redundant changes or changes that introduce inconsistencies at the level of the design of the target system. Second, there are no analyses to understand the dependencies between changes. The integrators are left to manually compare changes within the input stream of changes, and assess how these changes may impact the target system. Such work is particularly tedious between product forks, where the distance between branches grows larger over time.

In this paper we introduce a novel technique that tackles the above problems by modeling changes and the dependencies between them. Our technique, named *JET*, provides a first-class representation – based on the information contained in

* Corresponding author.

http://dx.doi.org/10.1016/j.scico.2014.07.012
0167-6423/

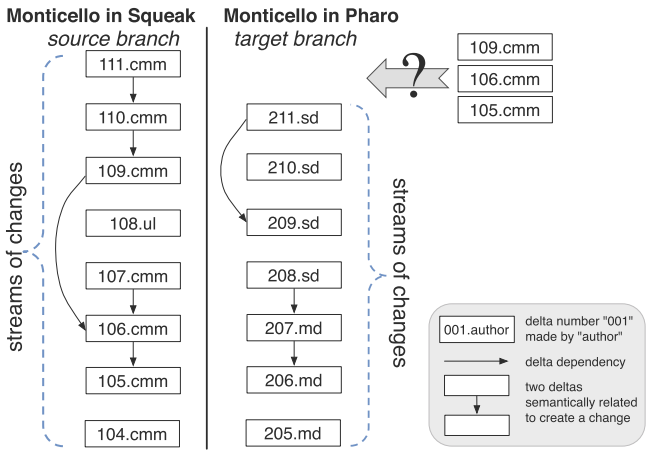

Fig. 1. Two branches of the Monticello versioning system and their stream of changes.

a version control system – of the history of the source code of a given system. By explicitly representing the changes between versions, and the dependencies between these changes, we provide additional information that guides integrators during the integration of changes. Such information is accessible for the integrators by means of simple queries (changes a certain change relies on, callers of a changed method) complemented by a dedicated dashboard and visualization that aid in comprehending sets of changes and the dependencies between such changes. We provide an implementation of our approach in Pharo.[1]

To illustrate our approach, we apply it to a concrete case study: the Squeak[2] forked versions of Monticello (a versioning system). We show how our approach aids in integrating forked versions of Monticello into the main distribution of Pharo. After forking, various components – such as Monticello – have evolved independently within Pharo and Squeak. We show how our tools aid in (a) cherry picking changes from this open-source project, (b) assessing the scale and impact of the changes, (c) determining which other changes these changes depend on, and (d) filtering irrelevant changes.

The contributions of this paper are: (1) A change and dependency model, as well as the algorithms for supporting streams of changes analyses. (2) A tool that provides lists of changes, deltas and dependencies of a stream of changes, along with a visual map of dependencies, and a browser to explore the history of any change within a given branch taking the dependencies between changes into account. (3) A qualitative assessment, in the context of a real-life open-source system, of our approach and tools.

2. Challenges in supporting merge operations

While merging tools support automatic merging of *textual* modifications to text files, the real challenge lies in taking into account the actual *contents* of the modifications during the merging process. The following example from Pharo/Squeak illustrates the problems faced daily by integrators that need to merge features in presence of change dependencies (by dependencies we mean that a change requires another one to achieve its purpose).

2.1. Task examples

Fig. 1 shows two streams (sequences) of changes in both branches of the Monticello core package. The integrator working on the target branch would like to understand the changes that have been performed in the source branch so that he can integrate some of the changes into the target branch.

Each node represents a *delta* (*i.e.*, a set of changes extracted from two versions). Note that there can exist dependencies between these deltas (indicated as directed edges), and that the numbers of the deltas in the *source* branch are unrelated to the numbers of the deltas in the *target* branch.

With current-day tool support, the integrator has to navigate the source branch *manually* to recover such dependencies between changes. Moreover, some part of the changes may conflict with the current target branch. Again these have to be identified manually. For a deep analysis of integrator tasks and needs refer to Chapter 3 of [22] which presents a full survey of integrator needs and questions. Several important tasks are summarized here as well.

Recover dependencies. The integrator has to navigate the source branch manually to recover the dependencies between the changes. As an example, consider the case in which an integrator wants to introduce the changes of the delta

[1] Pharo: http://www.pharo-project.org.
[2] Squeak: http://www.squeak.org.