Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development

Prasun Dewan
University of North Carolina
dewan@unc.edu

Rajesh Hegde
Microsoft Research
Rajesh.Hegde@microsoft.com

Abstract. Previous work has found that (a) when software is developed collaboratively, concurrent accesses to related pieces of code are made, and (b) when these accesses are coordinated asynchronously through a version control system, they result in increased defects because of conflicting concurrent changes. Previous findings also show that distance collaboration aggravates software-development problems and radical co-location reduces them. These results motivate a semi-synchronous distributed computer-supported model that allows programmers creating code asynchronously to synchronously collaborate with each other to detect and resolve potentially conflicting tasks before they have completed the tasks. We describe, illustrate, and evaluate a new model designed to meet these requirements. Our results show that the model can catch conflicts at editing time that would be expensive to manage at later times.

Introduction

Complex software must be developed collaboratively. While recently there has been some interest in synchronous pair programming, traditionally the collaboration is asynchronous, with programmers working independently on the same or different parts of the software. Even in pair programming, different pairs work asynchronously on the same project. In asynchronous software development, there is a need for coordination mechanisms to manage conflicts. Traditionally, such mechanisms are provided by version control systems, which require programmers to individually address the conflicts at check-in time. Inspired by the
findings that distance collaboration aggravates software-development problems (Herbsleb et al. 2000) and radical co-location reduces them (Teasley et al. 2000), we identify a new distributed computer-supported model of software development that provides semi-synchronous conflict-management in asynchronous software development. By conflict management we mean determining if there is a conflict, identifying how to resolve it, and performing the fix. By semi-synchronous collaboration we mean a mix of synchronous and asynchronous collaboration.

The general concept of breaking collaboration into diverging asynchronous and converging synchronous phases has been presented in previous work (Munson et al. 1994). Here we consider a concrete realization of this concept in which the synchronous phases are used only for conflict management. Lightweight system-provided mechanisms are used to make transitions between the two phases.

To investigate this and other ideas, we have extended the user interface of the Visual Studio software development environment – we call the extended user-interface CollabVS. The design, implementation, novelty, and all possible uses of the extensions are not a focus of this paper. In fact, some of the extensions are also provided by recent programming environments and can be easily improved. Here we focus on the narrower issue of the application of these mechanisms in developing a new conflict-management model.

The rest of the paper is organized as follows. We first derive the collaboration model based on the results of previous research. Next we present a simple but realistic joint programming example to illustrate the model. We then identify a joint software development task also designed to exercise the model that is more elaborate than the example but small enough to be carried out in a lab study. Next we present the actual study performed using the task, and end with conclusions and future work.

**Deriving the Model**

In this paper, we will talk about both previous work and our own contribution at the model level. A model abstracts out details of the user activities supported by a single tool or a set of integrated tools. As these activities are supported directly by the tool (set), we assume there are lightweight mechanisms to transition among them that do not require the use of the OS to explicitly start applications. By abstracting out tool details, it is easier to reason about them and improve their shortcomings. In fact, in this section, we will derive our conflict management model by identifying and refining the models supported by previous work on collaborative software engineering tools. Before we do so, let us first identify the problems these models address.

Brooks (1974) found that adding more people to a software team does not necessarily increase the productivity of the team because of coordination costs. This observation seems unintuitive for two reasons. First, documentation should re-