Integrating semantic interference detection into version management systems

Danhua Shao, Sarfraz Khurshid, and Dewayne E. Perry
Electrical and Computer Engineering, The University of Texas at Austin
{dshao, khurshid, perry}@ece.utexas.edu

Abstract

Global software developments intensify parallel changes. Although parallel changes can improve performance, their interferences contribute to faults. Current Software Configuration Management (SCM) systems can detect the interference between changes at textual level. However, our empirical study shows that, compared with textual interference, semantic approach is more effective and efficient in detecting interference in high-degree parallel changes. We propose to integrate semantic interference checking into SCM system. Semantic interferences detected during check in can alert developers to potential faults.

1. Introduction

In the development of global software projects, parallel changes are becoming increasingly prevalent. Multiple developers work on the same module or program at the same time. Parallel-style development is important because: development teams are distributed; the size of the software systems are often too large to be handled by few developers; and markets bring pressure to develop new features or new product quickly.

However, in a global development setting, parallel development also brings problems: 24-hour workdays shorten the interval between changes; geographic and temporal distances cause difficulty in coordination; and parallel changes in such environments are very likely to conflict with each other. A case study on a subsystem of Lucent Technologies 5ESS Telephone Switching System [1] shows that faults are strongly correlated with degrees of parallel change.

To detect interferences between parallel changes, current SCM systems use textual approaches. Conflicts are identified by matching the source code modified by one change with that modified by other changes during the same period.

However, our empirical study shows that, compared with the high density of faults found in high-degree parallel changes, the conflicts at the textual level are very low. According to the change history of 5ESS, only 3% of the changes made within 24 hours by different developers physically overlapped each others' changes.

To detect more conflicts for fault prediction and prevention, we proposed a semantic approach [3]. This approach is based on data dependency analysis and program slicing. Our empirical evaluation [2] shows that this semantic interference detection algorithm best detects interference in high-degree parallel changes. Thus, integrating semantic interference detection into current SCM systems can help developers to find more faults in global software development.

This paper describes our research on integrating a semantic interference detection algorithm with an SCM system.

2. Algorithm

Our semantic interference detection algorithm combines data dependency analysis and program slicing. Data dependency analysis discloses the semantic structure of a program. Program slicing can identify which semantic substructures are impacted by changes. The overlap of the impacted parts of the two changes is their interference.

Figure 1 illustrates the semantic interference detection algorithm. V0 is base version and V0→V1 and V0→V2 are two parallel changes on V0.

First, we identify data dependencies within each version. We use a triple (*var: def, use*) to represent a dependency, where *var* is the variable on which the dependence is built, *def* is the line that defines variable *var*, and the *use* line uses the variable defined at *def* line. For example, the dependencies in V0 are {(a: 1, 3), (b: 2, 4), (i: 3, 5), (j: 4, 5)}.

Second, identify the changed lines for each change. Change V0→V1 modified the first line from "a=0" to "a=1" while change V0→V2 modified the second line from "b=0" to "b=5".

Third, for each change, we calculate its semantic impact by forward slicing from the changed lines. Change V0→V1 modified Line 1. According to the variable defuse chains, {(a: 1, 3), (i: 3, 5)}, Line 3 and 5 are impacted. So the impact of change V0→V1 is {3, 5}. Similarly the impact of change V0→V2 is {4, 5}.

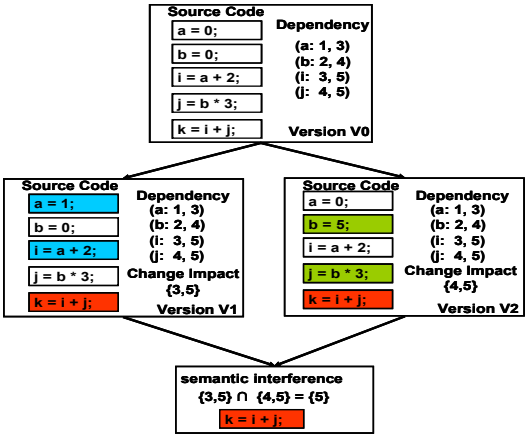

Figure 1. Detect Semantic Interference between V0→V1 and V0→V2.

Fourth, we identify semantic interference by checking the overlap between impacts of the two changes. The impact of change V0→V1 and impact of change V0→V2 overlap on Line 5, which is their semantic interference.

3. Implementation

To integrate semantic interference detection, an SCM system needs to change its repository and procedures for check-in and check-out.

- Repository: add a dependency graph for each version of every source file.
- Check-out procedure: record the base version number in local work space.
- Check-in procedure: add semantic interference detection to the existing textural interference detection

Figure 2 illustrates an example of the check-out and check-in procedure in a SCM with semantic interference checking. Given a version Vi, Vi.s represents the source code file and Vi.d represents the variable def-use dependency within version Vi. In Figure 2, V0 is the base version when a developer checks out a program file. After making changes on V0, the developer want to check in a new version Vc. At this moment, Vn is the latest version in repository. Figure 3 shows the steps of the check-in procedure.

4. Summary

Global software developments intensify the

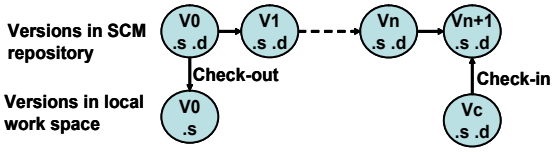

Figure 2. Version evolution in a SCM with semantic interference detection

```
1.   Check textual conflicts between
     V0.s→Vn.s and V0.s→Vc.s;
2.   Prompt developer to solve conflicts.
     Result is a new version Vc'.
3.   Check semantic conflicts between
     V0→Vn and V0→Vc' using
     dependencies in V0.d, Vn.d, and
     Vc'.d.
4.   Prompt developer to manually solve
     conflicts. Result is version Vc"
5.   if (Vc' = = Vc"){
        Save Vc".s as Vn+1.s;
        Save Vc".d as Vn+1.d;
        Done;
     } else {
        Vc = Vc";
        Goto step 1;
     }
```

Figure 3. The check-in process of a SCM with semantic interference detection.

interference among parallel changes. A 24-hour workday shortens the interval between changes. Geographic and temporal distances increase difficulties in coordination. Thus, detecting conflicts in parallel changes becomes important to project management and product quality.

We propose to integrate semantic interference detection into SCM systems because it is more effective than textual approaches in interference detection. With semantic interference detection, SCM can report more conflicts to developers when a new version is checked in. If a bug is reported, semantic impact and interference from previous versions can help developers to locate fault-inducing changes. Modules with high degrees of interferences can help development team to pay attention to the most dangerous parts. So SCM system with semantic interference detection is very helpful for developers to improve product quality in global software development processes.

Acknowledgements

This work was supported in part by NSF CISE Grant IIS-0438967.

Reference

[1] D.E. Perry, H.P. Siy, and L.G. Votta, "Parallel Changes in Large Scale Software Development: An Observational Case Study", *ACM Transactions on Software Engineering and Methodology*, Vol. 10, No. 3, July, 2001, 308-337.

[2] D. Shao, S. Khurshid, and D.E. Perry, "Evaluation of semantic interference detection in parallel changes: an exploratory experiment", *Proc. of the 23rd IEEE International Conference on Software Maintenance (ICSM'07)*, Paris, France, October 2007, 74-83.

[3] G.L. Thione, and D.E. Perry, "Parallel Changes: Detecting Semantic Interferences", *The 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, Edinburgh, Scotland, July 2005, 47-56.