SCA: a Semantic Conflict Analyzer for Parallel Changes

Danhua Shao Sarfraz Khurshid Dewayne E. Perry

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712, USA
{dshao,khurshid,perry}@ece.utexas.edu

Abstract

Parallel changes are becoming increasingly prevalent in the development of large scale software system. To further study the relationship between parallel changes and faults, we have designed and implemented a semantic conflict analyzer (SCA) to detect semantic interference between parallel changes. SCA combines data dependency analysis and program slicing. Data dependency analysis can disclose the semantic structure of the program. And program slicing can identify which semantic structures are impacted by a change. By comparing the overlap between impacts of two changes, SCA can detect if there are semantic interference between the two changes. An experiment with an industrial project shows that SCA can detect a significant portion of the faults in highly parallel changes. SCA is effective in predicting faults (based on "direct" semantic interference detection) in changes made within a short time period. SCA is both efficient (averaging less than two minutes) and scalable (requiring only the local context)

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Version control*; D.2.9 [**Software Engineering**]: Management – *Software configuration management*; D.3.3 [**Programming Languages**]: Language Constructs and Features – *Control structures*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages – *Program analysis*.

General Terms Languages, Management.

Keywords Version management, Semantic analysis, Data flow.

1. Introduction

Parallel development has become more and more important in the development of large-scale software systems. Multiple developers work on the same module or program at the same time. The need for parallel development has come about for a variety of reasons:

- the size of the software systems,
- time to market also brings pressure to develop new features or new products in a very short time,
- code ownership management is too expensive,
- the increase of globalization,
- and the geographical distribution of developers.

ESEC/FSE'09, August 24–28, 2009, Amsterdam, The Netherlands.
ACM 978-1-60558-001-2/09/08.

While parallel development increases productivity, it also causes problems. When developers work in parallel, it is likely that their changes may unintentionally interfere with each other. Our earlier research [3] showed the problems related to parallel changes. In a subsystem of Lucent Technologies' 5ESS Telephone Switching System, high degrees of parallelism happened at multiple levels. To disclose the relationship between parallel changes and faults, we studied *prima facie* conflicts at the textual level, checking the overlap between the lines changed by different developers. We found two important results: 1) 3% of the changes made with in 24 hours by different developers physically overlapped each others' changes; and 2) there was a linear correlation between the degree of parallelism and the likelihood of a defect resulting from the changes.

Given the small number of syntactic conflicts, we believe that conflicts at the semantic level are much more likely to be the cause of the resulting defects. To explore this hypothesis, we designed and implemented a semantic conflict analyzer (SCA) [4] to detect semantic interference using data dependency analysis and program slicing techniques [2]. To investigate the effectiveness of SCA, we designed a rigorous exploratory case study, and executed it in the same industrial context as our previous empirical studies.

2. Semantic Interference Detection in SCA

SCA uses data dependency analysis and program slicing to detect semantic interferences. With the data dependency analysis, we can learn the semantic structure of the program. And the program slicing can identify which semantic structures are impacted by changes. By comparing the overlap of the impacted parts of the two versions, SCA can identify if there are conflicts between them. The detailed explanation is in [5].

Figure 1 illustrates the semantic interference detection algorithm. Suppose there are two adjacent changes: v1→v2 and v2→v3.

1) For each version, calculate data dependence graph and identify variable def-use pair. 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 v1, the dependency is {(a: 1, 3), (b: 2, 4), (i: 3, 5), (j: 4, 5)}; version v2 is {(a: 1, 3), (b: 2, 4), (i: 3, 5), (j: 4, 5)}, and version v3 is {(a: 1, 3), (b: 2, 4), (i: 3, 5), (j: 4, 5)};
2) For each change, identify changed lines. In change v1→v2, Line 1 was changed and in change v2→v3, Line 2 was changed;

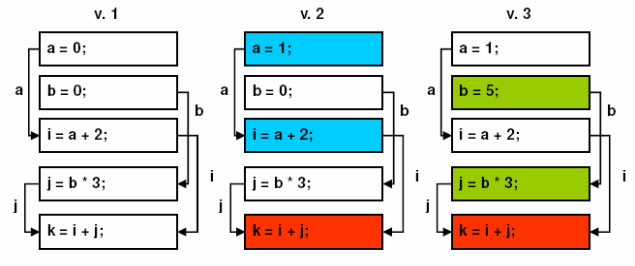

Figure 1 Detect semantic interference between changes: v1→ v2 and v2 → v3.

3) Calculate the semantic impact of the two changes by forward slicing from the changed lines. So, Impact (v1→v2) = {3, 5} and Impact (v2→v3) = {4, 5};
4) Compare impacted lines of the two changes. Line 5，where change (v1→v2) and (v2→v3) overlap with each other, is their semantic interferences.

Although change (v1→v2) and (v2→v3) are temporally sequential, they can also be viewed as logically parallel changes if they are made independent and committed by different developers within a short time. The change history used to evaluate SCA came from a pessimistic version control system. All changes were recorded sequentially because only one developer can check out and modify a program at a given time. The same approach can also be used with parallel versions in an optimistic version control systems.

3. Experiment and Results

To evaluate the effectiveness and efficiency of SCA, we conducted an experiment on the change and version management repository of one subsystem in 5ESS, a successful industrial project with high degree of parallel changes

3.1 Effectiveness Evaluation

To evaluate effectiveness of SCA, we prepared changes of differing degrees of parallelism. We constructed three sets of parallel changes from the change and version histories of 5ESS:

1) In the control set, interval between the versions are greater than 1 month, that they cannot be viewed as a parallel changes.
2) For the low degree of parallelism set, changes are made with a reasonable amount of interval time (from 1 week to 1 month). In this case, the developers have sufficient time to understand the implications of the changes made by others.
3) For the high degree of parallelism set, changes are made within a very short interval time, less than 1 week. In this case, it is difficult for the developers to understand the changes made by others in such a short time.

For each set of parallel versions, we run SCA to calculate the conflictions between changes, and calculate its *density of interference* (= detected interference/total versions). In control set, density of interference is 12%; in low degree set, density of interference is 29%, and in high degree set, density of interference is 49%. This result shows that SCA is effective in detecting semantic interference in high degree parallel changes.

To evaluate the effectiveness of SCA in fault prediction, we match between semantic interference detected by SCA with the actual faults in the fault sets for each changed version. The result shows that

1) SCA is effective in predicting non-pointer variable faults from interferences between changes made within short time periods and successfully predicted 32% of the existing faults.
2) Preciseness of pointer analysis, identification of variable renaming, and control-flow changes are the major factors that affect the effectiveness in detecting interference and predicting faults

3.2 Efficiency Evaluation

During the check on semantic interference, we also calculated the time to be saved if SCA is used to predict faulty code. The saved time will include two parts: the delay in fault detection and the elapse time in fault-fixing. From the change history of 5ESS, we learned that the average for fault-delay is 150 days, ranging from 59 to 262 days, and the average for fault-fixing elapse time is 3 days, ranging from 1 to 13 days.

But, compared with the times that could be saved, the overhead in calculating semantic interference is relatively very small: the average is about 2 minutes. In this overhead, 83% is spent on the program dependency analysis with CodeSurfer [1], and the time for the calculating and detecting interferences is even smaller than the time compiling the program.

4. Conclusion

Parallel development has become a common phenomenon in the development of large-scale software systems. Our earlier work delineated that there is a linear correlation between the degree of parallelism and the likelihood of a defect in the changes. However, textual analysis can only detect very small portion of change interferences. To detect change interference at the semantic level, we implemented a tool, SCA, which combining data dependency analysis and program slicing. Experiments with change history of 5ESS show that:

1) SCA detects a significant portion of the faults in highly parallel changes;
2) SCA is effective on non-pointer variable faults;
3) Compared with the time saved for fault detection and fix, the overhead of SCA is very low.

5. References

[1] GrammaTech, Inc. http://www.grammatech.com/
[2] S. Horwitz, J. Prins, and T. Reps, “Integrating non-interfering versions of programs”, *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 3, July 1989, pp 345-387
[3] 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, pp 308-337.
[4] 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.
[5] G.L. Thione, “Detecting Semantic Conflicts in Parallel Changes”, MSEE Thesis, The Department of Electrical and Computer Engineering, The University of Texas at Austin, December 2002. 98pp.