Chianti: A Change Impact Analysis Tool for Java Programs

Xiaoxia Ren, Barbara G. Ryder
Division of Computer and Information Sciences
Rutgers University
110 Frelinghuysen Road
Piscataway, NJ 08854, USA
xren,ryder@cs.rutgers.edu

Maximilian Stoerzer
Software Systems Department
University of Passau
94032 Passau, Germany
stoerzer@fmi.uni-passau.de

Frank Tip
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
tip@watson.ibm.com

ABSTRACT

Chianti is a change impact analysis tool for Java that is implemented in the context of the *Eclipse* environment. *Chianti* analyzes two versions of a Java program, decomposes their difference into a set of atomic changes, and a partial order inter-dependences of these changes is calculated. Change impact is then reported in terms of affected (regression or unit) tests whose execution behavior may have been modified by the applied changes. For each affected test, *Chianti* also determines a set of *affecting changes* that were responsible for the test's modified behavior. This latter step of isolating failure inducing changes for one specific test from irrelevant changes can be used as a debugging technique in situations where a test fails unexpectedly after a long editing session.

Catetories and Subject Descriptors: D.2.5 [Software Engineering]: Testing and Debugging; D.2.6 [Software Engineering]: Programming Environments; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages–*Program analysis*

General Terms: Algorithms, Measurement, Languages, Reliability

Keywords: Change impact analysis, regression test, unit test, analysis of object-oriented programs

1. INTRODUCTION

The extensive use of subtyping and dynamic dispatch in object-oriented programming languages makes it difficult to understand the flow of data through a program. For example, adding the creation of an object may affect the behavior of virtual method calls that are not lexically near the allocation site. Also, adding a new method definition that overrides an existing method can have a similar non-local effect. This *nonlocality of change impact* is qualitatively different and more important for object-oriented programs than for imperative ones.

Change impact analysis [1, 2, 4, 7, 3] consists of a collection of techniques for determining the effects of source code modifications, and can improve programmer productivity by: (i) allowing programmers to experiment with different edits, observe the code fragments that they affect, and use this information to determine which edit to select and/or how to augment test suites, (ii) reducing the amount of time and effort needed in running regression tests, by determining that some tests are guaranteed not to be affected by a given set of changes[1]. (iii) reducing the amount of time and effort spent in debugging, by determining a safe approximation of the changes responsible for a given test's failure [7, 6, 5].

The change impact analysis method presented in this paper presumes the existence of a suite $\mathcal{T}$ of regression tests associated with a Java program and access to the *original* and *edited* versions of the code. Our analysis comprises the following steps:

1. A source code edit is analyzed to obtain a set of *interdependent* atomic changes $\mathcal{A}$, whose granularity is (roughly) at the method level. These atomic changes include all possible effects of the edit on dynamic dispatch. Furthermore, a partial order between these atomic changes is determined, and this partial order captures dependences of changes that must be respected to create a syntactically valid program.

2. Then, a call graph is constructed for each test in $\mathcal{T}$. Our method can use either dynamic call graphs that have been obtained by tracing the execution of the tests, or static call graphs that have been constructed by a static analysis engine.

3. For a given set $\mathcal{T}$ of regression tests, the analysis determines a subset $\mathcal{T}'$ of $\mathcal{T}$ that is *potentially affected* by the changes in $\mathcal{A}$, by correlating the changes in $\mathcal{A}$ against the call graphs for the tests in $\mathcal{T}$ in the *original* version of the program.

4. Finally, for a given test $t_i \in \mathcal{T}'$, the analysis can determine a subset $\mathcal{A}'$ of $\mathcal{A}$ that contains all the changes that may have affected the behavior of t_i. This is accomplished by constructing a call graph for t_i in the *edited* version of the program, and correlating that call graph with the changes in $\mathcal{A}$.

Since the primary goal of our research is to provide programmers with tool support that can help them understand why a test is suddenly failing after a long editing session

[1]It is of benefit only if the cost of change impact analysis on a test case is less than the cost of rerunning the test.

ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.
ACM 1-58113-963-2/05/0005.

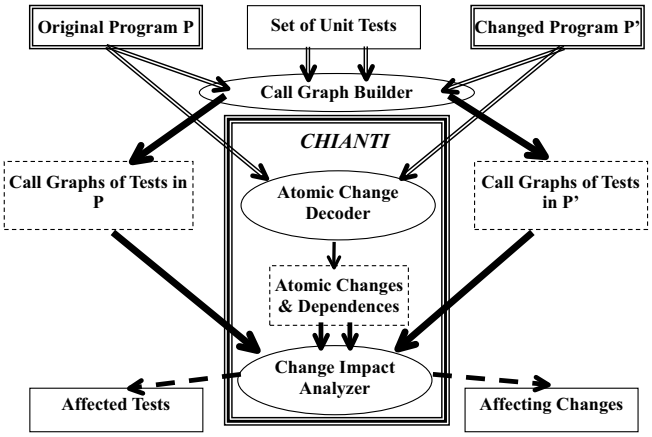

Figure 1: *Chianti* architecture.

by isolating the changes responsible for the failure, *Chianti* has been integrated closely with Eclipse, a widely used extensible open-source development environment for Java (see `www.eclipse.org`).

2. CHIANTI PROTOTYPE

Chianti is designed as an Eclipse *plugin*, which contributes a *launch configuration* and a *Chianti results view* to the Java perspective. *Chianti* conceptually can be divided into three functional parts. One part is responsible for deriving a set of atomic changes from two versions of a Java project, which is achieved via a pairwise comparison of the abstract syntax trees of the classes in the two project versions. Another part reads test call graphs for the original and edited projects, computes affected tests and their affecting changes. The third part manages the views that allow the user to visualize change impact information. *Chianti*'s GUI also includes a launch configuration that allows users to select the project versions to be analyzed, the set of tests associated with the project and the call graphs to be used. Figure 1 depicts *Chianti*'s architecture.

Although *Chianti* is intended for interactive use, we currently require that two versions of a program are saved in two separate Java projects, instead of comparing the current version with its local history. Thus, a typical scenario of a *Chianti* session begins with the programmer editing the current project, extracting the lastest stable version of this project from CVS repository into the workspace. The programmer then starts the change impact analysis launch configuration, and selects these two projects of interest as well as the test suite associated with these projects. Currently, we allow tests that have a separate `main()` routine and *JUnit* tests

In order to enable the reuse of analysis results, and to decouple the analysis from GUI-related tasks, both atomic change information and call graphs are stored as XML files. *Chianti* currently supports two mechanisms for obtaining the call graphs to be used in the analysis. When static call graphs are desired, *Chianti* invokes the *Gnosis* analysis engine[2] to construct them [6]. In this case, users need to supply some additional information relevant to the analysis engine (e.g., the choice of call graph construction algorithm to be used and some policy settings for dealing with reflection). Users can also point *Chianti* directly at an XML file representation of the call graphs that are to be used, in order to enable the use of call graphs that have been constructed by external tools. [5] presented the experiments with dynamic call graphs.

When the analysis results are available, a new view *Chianti results view* will stack on top of the outline view in the Java perspective. Since *Chianti* is expected to be part of programming and debugging, we integrate it into existing Java perspective rather than define a new perspective. The *Chianti results view* provides users two ways of traversing the analysis results.

- The *affecting changes view* shows all tests in a tree view. Each affected test can be expanded to show its set of *affecting changes*. Each affecting change is an atomic change that can be expanded on demand to show its prerequisite changes. This quickly provides an idea of the different "threads" of changes that have occurred.
- The *atomic-changes-by-category view* shows the different atomic changes grouped by category. The atomic changes and dependences shown in this view are not related to any specific tests, it just summarizes the results of comparison of two versions of a Java program.

Each of these user interface components is seamlessly integrated with the standard Java IDE in Eclipse (e.g., clicking on an atomic change in the *affecting changes view* opens an editor on the associated program fragment).

3. REFERENCES

[1] Shawn A. Bohner and Robert S. Arnold. An introduction to software change impact analysis. In Shawn A. Bohner and Robert S. Arnold, editors, *Software Change Impact Analysis*, pages 1–26. IEEE Computer Society Press, 1996.

[2] James Law and Gregg Rothermel. Whole program path-based dynamic impact analysis. In *Proc. of the International Conf. on Software Engineering*, pages 308–318, 2003.

[3] Alessandro Orso, Taweesup Apiwattanapong, James Law, Gregg Rothermel, and Mary Jean Harrold. An empirical comparison of dynamic impact analysis algorithms. In *Proc. of the International Conf. on Software Engineering (ICSE'04)*, pages 491–500, Edinburgh, Scotland, 2004.

[4] Alessandro Orso, Taweewup Apiwattanapong, and Mary Jean Harrold. Leveraging field data for impact analysis and regression testing. In *Proc. of European Software Engineering Conf. and ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE'03)*, Helsinki, Finland, September 2003.

[5] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. Chianti: A tool for change impact analysis of Java programs. In *Proc. of the Conf. on Object-Oriented Programming, Systems, Languages and Applications*, Oct 2004.

[6] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, Ophelia Chesley, and Julian Dolby. Chianti: A prototype change impact analysis tool for Java. Technical Report DCS-TR-533, Rutgers University Department of Computer Science, September 2003.

[7] Barbara G. Ryder and Frank Tip. Change impact for object oriented programs. In *Proc. of the ACM SIGPLAN/SIGSOFT Workshop on Program Analysis and Software Testing (PASTE01)*, June 2001.

[2] *Gnosis* is a static analysis framework that has been developed at IBM Research as a test-bed for research on demand-driven and context-sensitive static analysis.