Automated Testing of Eclipse and NetBeans Refactoring Tools[*]

Brett Daniel Danny Dig Kely Garcia Darko Marinov
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{bdaniel3, dig, kgarcia2, marinov}@cs.uiuc.edu

ABSTRACT

This position paper presents our experience in automated testing of Eclipse and NetBeans refactoring tools. Test inputs for these tools are Java programs. We have developed ASTGen, a framework for automated generation of abstract syntax trees (ASTs) that represent Java programs. ASTGen allows developers to write *imperative generators* whose executions produce ASTs. ASTGen offers a library of generic, reusable, and composable generators that make it relatively easy to build more complex generators. We have developed about a dozen of complex generators and applied them to test at least six refactorings in each tool. So far, we have found 28 unique, new bugs and reported them, 13 in Eclipse Bugzilla and 15 in NetBeans Issuezilla. This is ongoing work, and the numbers are increasing.

We advocate the importance of automated testing—not only automated execution of manually written tests (using JUnit or XTest) but also *automated generation of test inputs*. We have developed several oracles that programmatically check whether a refactoring tool correctly made some program transformations (or gave warning that a specific refactoring should not apply to the given input program).

We hope that this paper motivates developers of refactoring tools to incorporate such generation and oracles into their tools. While most refactoring tools are already quite reliable, we believe that the use of such generation would further increase reliability, to the benefit of all users of refactoring tools. Moreover, we argue that such generation can be useful for testing other related tools that take (Java) programs as inputs. To encourage collaboration and enable others to try out ASTGen, we have made our ASTGen code and all experimental results publicly available at the ASTGen web page, `http://mir.cs.uiuc.edu/astgen`

1. WHY AUTOMATED GENERATION?

Testing involves several activities, including generation of test inputs (and expected outputs), execution of test inputs, and checking of obtained outputs. For a refactoring tool, each input consists of a program and a refactoring to apply, and each output is either a refactored program or a warning if the specific transformation might change the program's semantics.

[*]This paper is based on the work [1] to be presented at the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007) in Dubrovnik, Croatia.

It is often said that manual testing is tedious and error-prone. Indeed, developers of refactoring tools automate a large portion of testing. For instance, we have counted 2,673 JUnit tests for the major refactorings in Eclipse version 3.2. JUnit automatically executes these tests and checks the obtained outputs. However, JUnit does not automatically generate test inputs, and to the best of our knowledge, Eclipse developers manually wrote their JUnit tests.

Automated generation of test inputs has one significant benefit: it makes it easier to generate a large number of test inputs, which hopefully results in a more thorough testing and enables finding bugs before they are encountered in production runs. However, automated generation of test inputs, especially for refactoring tools, poses several challenges. We discuss these challenges and our solution.

1.1 Generation of Input Programs

How does one automatically generate valid Java programs to give as inputs to a refactoring tool? There is no obvious answer. While simpler test inputs, say one integer or a sequence of integers, can be generated even randomly, it is unclear how one could randomly generate sequence of characters (or abstract syntax trees) that satisfy the syntactic and semantic constraints for a valid Java program. Moreover, even if one could generate programs randomly, how would one ensure that these programs have properties relevant to the refactoring under test?

We have developed the ASTGen framework to generate a large number of relevant Java programs. ASTGen is not fully automatic. It requires developer to write a class (which we call *generator*) that can produce test inputs relevant to a specific refactoring. ASTGen provides a library for generation of simple AST nodes. This library makes it easy to build more complex combinations of AST nodes. We have written several generators that produce Java programs for testing refactoring engines. More details are in the conference paper [1]. We point out that the generators do not always produce programs that compile. (The column "CI" in Figure 1 shows how many of the generated inputs compile and are thus valid inputs for a refactoring.)

1.2 Execution of Refactorings

How does one automatically run a refactoring tool on the automatically generated input programs? This is seemingly an easy task: just develop a piece of code that (efficiently) runs a specific refactoring on each of the generated programs. However, we have encountered a number of problems while developing this piece of code, in both Eclipse and

	Generation				Oracles						**Bug Reports**	
Refactoring	Generator	TGI	Time [min:sec]	CI	WS		DNC		C/I	Diff		
					Ecl	NB	Ecl	NB			Ecl	NB
Rename(Method)	MethodReference	9540	89:12	9540	3816	0	0	0	0	5724	0	0
Rename(Field)	FieldReference	3960	28:20	1512	0	0	0	304	0	40	0	1
EncapsulateField	ClassArrayField	72	0:45	72	0	0	48	0	0	48	1	0
	FieldReference	3960	15:19	1512	0	0	320	432	14	121	4	3
	DoubleClassFieldRef.	14850	41:45	3969	0	0	187	256	100	511	1	2
	DoubleClassGetterSetter	576	8:45	417	216	0	162	162	18	216	2	2
PushDownField	DoubleClassFieldRef.	4635	10:56	1064	760	380	152	228	0	380	2	2
	DoubleClassParentDecl.	360	6:50	270	246	168	18	90	0	78	1	2
PullUpField	DoubleClassChildDecl.	60	1:14	44	0	18	10	6	0	44	1	1
MemberToTop	ClassRelationships	70	0:36	51	0	0	0	2	0	2	0	1
	DoubleClassFieldRef.	6600	29:04	2824	0	0	353	507	0	2824	1	1
										Total Bugs:	**13**	**15**

Figure 1: Refactorings tested and bugs reported, Ecl = Eclipse, NB = NetBeans TGI = Total Generated Inputs, Time in [min:sec], CI = Compilable Inputs, WS = WarningStatus, DNC = DoesNotCompile, C/I = Custom/Inverse, Diff. = Differential

NetBeans. These problems have been partly due to certain design decisions in these refactoring tools.

Two key problems that we encountered were (1) how to reduce the dependency of the refactoring under test from the rest of the IDE and (2) how to efficiently execute the refactorings. We still have not solved the first problem satisfactorily in Eclipse. Namely, our testing of refactorings requires that we run Eclipse in the GUI mode, which not only slows down the execution but also disallows using (fast) servers with a text-only connection. We still have not solved the second problem satisfactorily in NetBeans. Namely, our testing does not release all the resources after each refactoring. (Specifically, it creates a new project for each input program.) This results in an increasing memory usage over time and requires that we rerun NetBeans several times, splitting a large number of input programs into several smaller batches that can each fit into one run. We hope that developers of refactoring tools can provide better "hooks" for running automatically generated inputs programs.

1.3 Checking of Outputs

How does one automatically check the outputs that a refactoring tool produces for the automatically generated inputs? While this problem is related to checking correctness of compilers [2] (the output program should be semantically equivalent to the input program), in addition, the refactored program should have the intended changes.

We have developed a variety of oracles for programmatic checking of refactoring tools. The simplest oracle checks that the refactoring tool does not throw an uncaught exception, but we have not found such a case in either Eclipse or NetBeans. The WarningStatus (WS) oracle checks whether the tool produces a warning or a refactored program. The DoesNotCompile (DNC) oracle checks whether the refactored program compiles. The Custom/Invertible (C/I) oracle checks specific structural properties (e.g., moving an entity should indeed create the entity in the new location) or invertibilty (e.g., renaming an entity from A to B and then from B to A should produce the same starting input program). The Differential (Diff) oracle [2] gives the same input program to both Eclipse and NetBeans and compares whether they produce the same output.

1.4 Experimental Results

When is testing with automatically generated inputs applicable? There are at least two benefits of manually written tests. First, in test-driven development, the tests are written even before writing the code, and thus such tests help in designing the code. Second, in regression testing, when developers want to get a quick feedback about the code changes they are making, it is better to use a smaller number of tests manually written (or previously manually selected from some automatically generated tests) than to use a large number of automatically generated tests. However, we claim that when developers can run tests for longer time or want to exercise their code more thoroughly, it is appropriate to use automatically generated tests (in addition to manually written tests).

Figure 1 shows some of our experimental results that support the above claim. (The full results are available in the conference paper [1].) The "Time" column shows the total time required to generate the input programs and to run them in Eclipse. Running is over an order of magnitude slower than generation. Testing each refactoring takes less than an hour and a half (on a a dual-processor 1.8 Ghz machine), and the entire suite can be run overnight. The benefit is finding new bugs, as shown by a total of 28 new bugs in Eclipse and NetBeans.

2. BEYOND REFACTORING TOOLS

We believe that automated testing based on ASTGen generators is useful beyond refactoring tools. In principle, any tool that operates on programs (or abstract syntax trees) could benefit from ASTGen. The main question is how easy/hard it is to write the generators that produce interesting programs that satisfy the required constraints. We plan to investigate this in new application domains, e.g., in other parts of Eclipse and NetBeans IDEs.

3. REFERENCES

[1] B. Daniel, D. Dig, K. Garcia, and D. Marinov. Automated testing of refactoring engines. In *ESEC/FSE 2007*, Dubrovnik, Croatia, Sept. 2007. (To appear.).

[2] W. M. McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1), 1998.