Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2011-IC-002

Feedback-directed test case generation based on UML activity diagrams

Xin Chen, Nan Ye, Peng Jiang, Lei Bu, Xuandong Li

Postprint Version. Originally Published in: International Conference on Secure Software Integration and Reliability Improvement-Companion 2011, IEEE Computer Society Press

Feedback-directed test case generation based on UML activity diagrams

Xin Chen, Nan Ye, Peng Jiang, Lei Bu, Xuandong Li
State Key Laboratory for Novel Software Technology, Nanjing University
Department of Computer Science and Technology, Nanjing University
Nanjing, Jiangsu, P.R.China 210093
Email: chenxin@nju.edu.cn,{yenan,jiangp}@seg.nju.edu.cn,{bulei,lxd}@nju.edu.cn

***Abstract*—As UML models are widely used as design blueprints, model-based techniques can be used in test case generation. However, test cases obtained from these techniques are usually abstract test cases, represented as sequences of actions in abstract models, and heavy human efforts are needed to translate them into concrete test cases accepted by programs for execution. To reduce this effort, we present an approach to automatically generating executable test cases based on activity diagrams. It relates methods of classes in JAVA programs with activity nodes in their design models and instruments codes into JAVA programs to collect traces in execution. Regarding traces collected in execution as feedbacks, data classifiers that can predict test inputs' impact on decision nodes in activity diagrams are constructed. Those data classifiers are used to guide the creation of new test inputs that can cover untouched paths in design models. Experiments show that the approach can greatly relieve testers' burden in preparing test cases.**

***Keywords*-Test case generation; Feedback-directed; Activity Diagram;**

I. Introduction

Software testing is the most widely used approach to ensuring the quality of software. As the complexity and size of software systems grow up quickly, manual testing becomes so labor-intensive and error-prone that it is worth developing automatic testing techniques, such as automatic test case generation.

Model-based approaches are widely used in test case generation. However, they still require heavy manual efforts. Test cases obtained from model-based approaches are sequences of actions in the model, which can not be directly fed to the software under test. For each sequence of actions in the model, testers need to work out inputs that can drive the software's execution to cover such a sequence. Manually translating abstract test cases into concrete ones needs heavily human efforts and a lot of time. Thus, how to automatically generate concrete test cases from design models becomes an important problem worthy to be studied.

This paper presents an approach to automatically generating test inputs for JAVA programs with respect to the simple path coverage adequacy criteria of their design models i.e. activity diagrams. A simple path refers to a feasible execution in an activity diagram that starts from the initial node and end with final nodes. The word 'simple' means a path contains no circles. With the help of testers, methods of classes in programs are related with their counterparts, activity nodes in activity diagrams. Based on the relation, tracing codes are instrumented into programs. Then, instrumented programs are executed with generated test inputs. Outputs generated by instrumented codes can show the running traces of methods of classes. By collecting these feedbacks from execution, the approach constructs a classifier for each decision node in activity diagrams. Each classifier can predict test inputs' impact on its corresponding decision node. Thus for any uncovered simple path, those classifiers can point out what should the required test inputs look like.

Different from existing works[2] which can only generate abstract test cases. This approach can automatically generate concrete test inputs accepted by programs. Compared with the relevant work in [3], the classifiers used to guide test case generation are not constructed by testers manually but generated by machines automatically. This approach is also more effective than our previous work in [1], where the random test case generation method is applied.

II. Feed-back directed test case generation

The workflow of our approach is shown in Figure 1. It consists of three phrases:

1) The **preparation before execution** phrase consists of two sequential steps: *Relate methods of classes with activities in activity diagrams* and *Instrument JAVA programs under test based on the matching configuration*. On most occasions, the relating process can be carried out automatically by matching names of methods and classes on both sides. Methods and classes on both sides usually share the same names since classes in implementation must conform to their counterparts in design models. Unmatched classes and decision nodes are related by testers. The tracing codes are instrumented at two places. Before any interested method is invoked, a piece of codes that can print the caller's object identifier, the caller's type and the current thread's identifier is inserted. At the same time,

This work is supported by the National Natural Science Foundation of China (No.91018006, No.61003025), National S&T Major Project (2009z01036-001-001-3), and by the Jiangsu Province Research Foundation(BK2010170).

978-0-7695-4454-0/11 $26.00 © 2011 IEEE
DOI 10.1109/SSIRI-C.2011.12

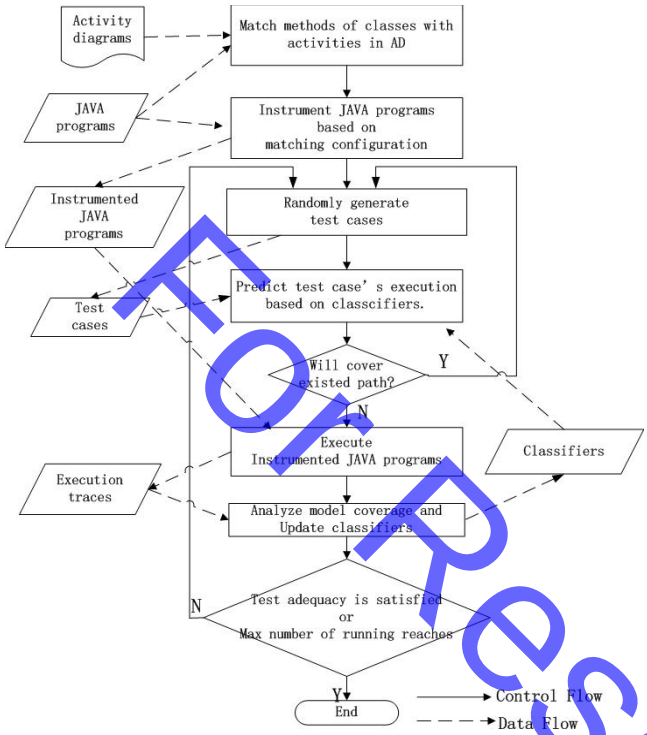

Figure 1. The Workflow of FDT method

a piece of codes printing the callee's object identifier, the callee's type and the current thread's identifier is inserted before the first statement in any interested method's body.

2) The **primary data classifiers generation** process includes threes steps. First, a number of test inputs are randomly generated. Then, the instrumented program is executed and fed with these inputs. At last, when the program terminates, traces outputted in execution are collected and analyzed. As JAVA programs permit classes to be shared by many threads, thread identifiers help to distinguish method's invocation coming from different threads. Object identifiers help to match callees with their callers. Type information helps to identifier callee's full name. Orders in output traces give the temporal orders of methods' executions. All these information are gathered to construct data classifiers of decision nodes in activity diagrams.
3) Before a randomly generated test case is executed, its execution path in the activity diagram will be predicted by calculating data classifiers of decision nodes. Useless inputs that are unable to raise path coverage will be dropped. To raise their accuracy, data classifiers are updated after each test case is executed.

III. Experiments

An online stock exchange system(OSES) is used in the experiment. It is a JAVA program, reconstructed from an example in [4] which consists of 40 classes and 305 methods. The activity diagram, acting as its design model, has 25 activities, 12 decision nodes and 18 paths.

Table I
THE PERFORMANCE OF CDT

	test cases		cov paths	cov rate	time
	total	exec			
CDT	102453	100	14.2	0.789	102s
CDT	208219	200	15.6	0.867	208s
RT	100	100	13.2	0.733	98s
RT	200	200	15	0.833	199s

We set the max number of execution to 100 and 200. For data Classifier Directed Test case generation(CDT), half of the executed cases are used to build data classifiers. Each experiment is repeated five times and the average of path coverage results are recorded in Table I.

The table shows with the help of data classifiers, the CDT method achieves higher path coverage with the suite of the same size. By preventing similar test cases from execution, the CDT method can travel wider area of the input domain. At the same time, the computation of classifiers does not bring too much overhead to the time. When we set the max number of execution to 1000, both methods can achieve 100% path coverage. And on most occasions, the test suite generated by CDT is smaller than that by Random Test case generation(RT).

IV. Conclusion

In this paper, we have proposed an automatic test case generation approach based on activity diagrams. It uses runtime behaviors of programs as feedback to pick up useful test cases with respect to the simple path coverage criterion in activities diagrams from test cases generated in random. Experiments show the approach can greatly relieve testers' burden in preparing test cases.

References

[1] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, X. LI. UML Activity Diagram-Based Automatic Test Case Generation For Java Programs. The Computer Journal, Vol.52, No.5, Oxford Press, 2009, pp.545-556.

[2] L. C. Briand, Y. Labiche, K. Buist, G. Soccar. Automating impact analysis and regression test selection based on UML designs. In proceedings of the International Conference on Software Maintenance (ICSM'02), IEEE Computer Society, 2002, pp 252-261.

[3] A. Krupp and W. Mueller. Classification trees for random tests and function coverage. In proceedings of the conference on Design, automation and test in Europe (DATE'06). European Design and Automation Association. 2006. pp 1031-1032.

[4] M. Blaha, J. Rumbaugh. *Object-Oriented Modeling and Design with UML(Second Edition)*. Pearson Education Inc. 2005.