A Tool for integrated Test Driven Development – iTDD

Wolfgang Schwaiger
UAS Northwestern Switzerland
Institute for Mobile and Distributed Systems
5210 Windisch, Switzerland
wolfgang.schwaiger@fhnw.ch

Martin Kropp
UAS Northwestern Switzerland
Institute for Mobile and Distributed Systems
5210 Windisch, Switzerland
martin.kropp@fhnw.ch

Abstract

While testing is more and more an established practice in software development, writing and maintaining tests still causes a significant effort. This paper presents a concept for an integrated testing tool which will relieve the developer from writing test code manually and help to reduce the overall effort for test implementation significantly. The tool is based on the concept of a strict separation of test data and test code which minimizes test code on one side and eases test data specification on the other side. Our goal is to provide the developer with a tool for fully automated test generation and maintenance. When changing the production code the tool will be able to refactor the corresponding test code and test data based on the given information and eliminates the need for manual modification. We believe that the proposed tool will help to increase overall testing efficiency significantly and finally lead to better software.

1. Introduction

Although writing test cases, especially on unit level, becomes more and more popular and an accepted development method, many developers are still reluctant to write tests [4]. Reasons for this are a) writing "good" tests is still hard and causes a significant effort [9]; b) maintenance of the test suites is time-consuming and occupies resources needed in the development [2, 5]; c) the intertwining of test code with test data makes it difficult to specify new and extend existing test data since each new test data set requires either new test code or changes in the source code [2, 5]; d) users are prevented from writing concrete test specifications on acceptance level since the test data is in the code [8]. While automated test data generation has been addressed in many research papers (e.g. [3, 6, 7]), automated maintenance of test code and test data is still an open issue (see [2, 5]) which will be the focus of our work.

2. An iTDD Tool

To address the aspects under b), c), and d), we follow the strict separation of test code and test data as it is suggested in [8, 10]. This approach offers the following advantages – test data can be entered separately in a more user centric interface (e.g. tables) – it minimizes the test code, since test code acts like an intermediary between the test data and the system under test (SUT). This allows to call the same test code for a set of test data of the same kind. With this separation our tool aims to generate the complete test code implementation based on a given method signature and also to provide automatic test code and test data refactoring. The now completely separated test data allows to focus on the more important aspect of test data specification on acceptance level. The tool will be integrated into a development environment to allow maximum efficiency for the developer.

2.1. Addressed Use Cases

One typical use case scenario in Test Driven Development (TDD) may start with the implementation of a new class. When a new method has been specified, the test code implementation and the related test data will be generated automatically. The developer can add new or edit existing test data in a tabular form from within the IDE (e.g. Eclipse). The tests can already be executed but will fail as long as the method implementation is not complete. In the case of adding tests to legacy code the developer just selects the desired method for test case generation. Moreover, during maintenance as the developer refactors a method the tool will automatically update the test data and the test code.

2.2. Test Generation

Based on the given method signature the tool will generate the test code and a test data repository. As we fo-

cus on the maintenance of test code and test data we will generate sample input data for the data repository. The test data repository will be realized in a test engine independent XML format. From their the test data will be converted into a specific test engine format. This enables to support various test engines which follow the concept of test code and test data separation like the Fit framework [8] and Concordion [10].
In a first step we will target the Fit framework as the test engine for the iTDD tool. This means that the tool will automatically generate and maintain the Fit fixture code. The mapping of the signature of the method under test (MUT) and the test data is done by convention. The names of the column headers of the test data tables correspond with the names of the parameters of the method signature.

2.3. Test Maintenance

To enable full test code maintenance our tool will apply the mapping technique as aforementioned to track changes of the MUTs. This includes refactoring actions like to rename, add, delete, and move of parameters and methods. Test data and test code is automatically carried along these changes and the tool updates all impact test code and test tables.

2.4. Identified challenges

In [1] A. Bertolino lists several fundamental challenges in the field of software testing, like test effectiveness, test input generation, and test oracles. For our work we identified the following additional challenges: a) generation of test code for testing search results, queries, lists, sets, or bag of things; b) generation of test code for testing sequences of actions made to a system; c) to support test data generation of user-defined data types, collections, structures, etc.; d) keep test data and test code in sync with the refactored production code.

3. Current and Further Work

3.1. Current Work

To validate the presented concept and prove the applicability of the iTDD tool we currently implement the test case generation for new code, for legacy code, as well as for the refactoring of existing code using the Fit test engine. In a first step we target functional tests with basic data types like string, integer, and double for the Fit ColumnFixture fixture type. In a next step we will include support for classes, self-defined data types, and collections.

3.2. Hypothesis and Evaluation

We believe that this approach can provide a fully automated generation and maintenance of test code and test data without the need for manual test code modifications. Thus making a further step towards "100% test automation" as described in [1]. In order to evaluate the proposed technique and tool the empirical body of evidence will be given by the ProMedServices Project (`www.fhnw.ch/technik/projekte/promedservices`). It will be used to collect data and sets a real world context in industry to the appliance of our tool. This includes the range of supported data types and the number of supported refactoring methods.

3.3. Further Work

Further work will cover the development and integration of test data generation based on contract information such as given by Contract4j (`www.contract4j.org`) or JML (`www.jmlspecs.org`).

References

[1] A. Bertolino. Software Testing Research: Achievements, Challenges, Dreams. In *International Conference on Software Engineering*, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[2] A. Deursen, L. Moonen, A. Bergh, and G. Kok. Refactoring test code. In M. Marchesi and G. Succi, editors, *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001)*, May 2001.

[3] J. Edvardsson. A survey on automatic test data generation. In *Proceedings of the Second Conference on Computer Science and Engineering in Linköping*, pages 21–28, Oct. 1999.

[4] B. George and L. Williams. An Initial Investigation of Test Driven Development in Industry. In *SAC*, Melbourne, Florida USA, 2003.

[5] E. M. Guerra and C. T. Fernandes. Refactoring Test Code Safely. In *Proceedings of the International Conference on Software Engineering Advances(ICSEA 2007)*, 2007.

[6] A. Milicevic, S. Misailovic, D. Marinov, and S. Khurshid. Korat: A Tool for Generating Structurally Complex Test Inputs. *Formal Research Demo at the 29th International Conference on Software Engineering (ICSE Demo 2007)*, May 2007. Minneapolis, MN.

[7] M.Prasanna, S. Sivanandam, R.Venkatesan, and R.Sundarrajan. Survey On Automatic Test Case Generation. *Academic Open Internet Journal*, 15, 2005.

[8] R. Mugridge and W. Cunningham. *Fit for Developing Software*. Pearson Education, Inc., New Jersey, 2005.

[9] S. Nageswaran. Test Effort Estimation Using Use Case Points. In *Quality Week 2001, San Francisco, California, USA*. Cognizant Technology Solutions, June 2001.

[10] D. Peterson. Concordion. `http://www.concordion.org/` (07.07.2008), 2007–2008.