Test Composition with Example Objects and Example Methods

Markus Gälli

Institut für Informatik und Angewandte Mathematik
University of Bern, Switzerland

IAM-03-009

February 17, 2004

Abstract

While assertions of *Design by Contract* from Eiffel [7] found their way into the language-definitions of Python and of Java SDK 1.4, current object-oriented languages do not make the concepts of *test first programming*[2] explicit in their definitions or meta-models. Not having support of unit-testing in a programming language makes it harder to compose and re-compose test-scenarios and tests. I propose, that an object-oriented language should include explicit concepts for example objects, example methods and instance-specific assertions. This concepts ease the composition of complex test-scenarios, they help to refactor the program with the tests and also to keep the duration of the tests as low and the coverage of the tests as high as possible.

CR Categories and Subject Descriptors: D.2.5 [Software Engineering]: Testing and Debugging—*Testing tools*; D.3.3 [Programming Languages]: Language Constructs and Features—*Classes and objects*;

1 Introduction

Automatic unit-tests and the concept of *Design by Contract* are crucial in software development. Though unit-tests have proved their usefulness, they don't provide means to refactor tests together with the program or to compose test-scenarios or tests. With assertions of design by contract on the other hand, it is easier to refactor the program and to check several contracts in one execution path, but these contracts start on an abstract level and are not automatically executable. As unit-tests can be seen as contracts of individual methods on individual objects, we will introduce a blend of example objects and example methods to enable individual contracts and show, how this approach eases test composition and re-composition.

We concentrate on problems which occur while testing, as the bad smells, which arise there, could show us some problems of current class-based object-oriented languages.

1.1 Problem: Test Refactoring

Tests have to be refactored, when the program is refactored, otherwise their breaking only shows the absence of maintenance and not the presence of bugs.[4] Assertions of design by contract do not care, if their method is renamed or moved to another place, they will be renamed or moved automatically with the method. So they are local to the method, even if the method breaks. This is not the case with unit-tests, if the method under test is moved to another class, the programmer has to move the test-method itself manually to another test-class, at least if the style of "one class under test belongs to one testclass" is followed. Also renaming hurts, as the information, which method a test-method tests, is encoded in the name of the test-method, which won't be affected by renaming the method under test.

1.2 Problem: Scenario composition

Test-scenarios should be composable, so that the tester can build more complex scenarios out of simpler ones. Smalltalk-developers could create factory-methods on the class-side of the object using class-extensions, this factory methods recreate the examples, thus testing-code can be stored in an extra package and examples are composable. But taking Squeak as an example, this style is not yet followed, partially because Squeak is still lacking package support, partially because having examples is only a voluntary convention. So it is hard to setup complex test-scenarios for Squeak-objects, as even simple classes lack examples. Java has no capability of method-extensions, one solution is to use an extra factory-class, which is responsible for setting up the test-scenarios.[8]

This conventions and frameworks show, that current languages miss an explicit and possibly enforced way to store and recreate example objects.

1.3 Problem: Assertion composition

As any interesting method should be accompanied by a unit-test, a method under test can be very technical and low level but also represent a whole use-case. When a test-driven developer builds a system from the bottom up, lots of the execution-paths of the tests will overlap, so while testing higher level methods, lower level methods will be executed, that already had their own tests. As a consequence, the test-driven developer either has to throw away the low level tests, loosing some test-information, which could help him later to localize some bugs, or he has to wait longer for some feedback.

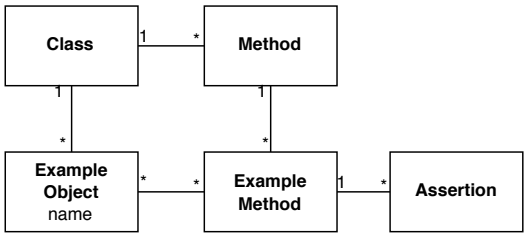

Figure 1: Object-diagram of example objects and example methods

2 Approach

2.1 Example objects and example methods

An example object is responsible for making an object reproducible by giving it a name. The programmer can save an object under a name and ask its class later to reproduce the object named like that. Example objects can also be renamed and deleted and they are the building-blocks for more complex example objects. An example method is a method, which has an example object as a receiver and example objects as parameters. It is always executable. Each time, an example object is instantiated, it registers itself in all example methods, in which it is used either as a receiver or as a parameter.

2.2 A prototype for example-composing: Eg

We implemented a prototype called *Eg* in Squeak [6]. *Eg* enables the programmer to store objects as named example objects, to visually compose example objects by dragging example objects into empty variables of other example objects, to visually create example methods by dragging methods on example objects (and eventually example objects into the parameters of the example method), to annotate pre- and post-conditions to that example methods and to execute and debug example methods.

Having explicit example objects, example methods and relations between them, we can simulate instance-specific assertions: When an example object is instantiated, we ensure, that a method-wrapper [3] is installed on each method, which has an example method and collaborates with this example object, either using it as a receiver or as a parameter. Then we register the example object in example method of these wrappers.

When such a method is called, the wrapper first checks, if the actual parameters are equal to any of an installed example method, if yes, the example method is triggered and its assertions are executed.

If an example object is reused in a more complex scenario, the example methods of this object will thus be reused and the stand alone test be made obsolete.

2.3 Validation

- Tests are automatically moved to the new class, when the method is moved to the new class
- Tests can be composed, so that each test is only executed once

- Example objects serve as documentation for possible usage of the class
- Tests don't have to break data-hiding, as instance-specific assertions can read any variable of the instance[1]
- Example objects could be used for reproducible type-reconstruction, thus making static typing obsolete
  - The IDE could provide tools like code-completion etc. known from statically typed languages
  - Performance could be optimized early by inlining machine-code

3 Discussion

Possible research-tracks include:

- How should a class-based language integrate method-examples cleanly? Does it make sense to introduce instance-specific methods[2][5][1] in general or only tailored to this testing-reasons?
- Replace the need to give a type in statically typed languages, where otherwise the compiler would complain, with the need to give an example in a dynamically typed language, where otherwise a testcase would fail. Would developers accept to be enforced to provide at least one example when otherwise they get a failing testcase?
- Use example objects for reproducible type-reconstruction,
- Use the fact, that 83% of all instance-methods in Squeak are unary-methods, apply them or a sample of them to well formed example objects and see, which test coverage you can get. Insert in them instance-based assertions. Research the percentage of unary methods in other languages.
- Example objects and example methods stand between prototypical systems and class-based systems, in example-oriented programming, you start with examples, thus prototypes, but the name example infers that you will generalize your example immediately. Abstractions or classes in class-based object-oriented systems have to be build before a concrete example-scenario for that class can be given. On the other hand in prototype-languages you are not forced to give an abstraction at all, which also might be problematic. Building classes by first composing examples, giving that composed example a name and only then a class-name could be an interesting way in between prototypical and class-based languages.

[1] *Eg* currently implements instance-based assertions with blocks, this feature could be implemented using some form of instance-specific methods

[2] Ruby implements instance-specific methods and calls them singleton-methods[9]

4 Conclusion

Object-oriented environments should allow developers to compose example-scenarios to more complex scenarios and tests to more complex tests. We proposed to accomplish this using example objects and example methods. We offered some validations of the solution and gave an outlook about possible research-tracks of using and enforcing examples like reproducible type reconstruction.

Acknowledgments

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the following projects:

- "Tools and Techniques for Decomposing and Composing Software" (SNF Project No. 2000-067855.02, Oct. 2002 - Sept. 2004).

References

[1] K. Beck. Instance specific behavior: How and Why. *Smalltalk Report*, 2(7), May 1993.
[2] K. Beck. *Test Driven Development: By Example*. Addison-Wesley, 2003.
[3] J. Brant, B. Foote, R. Johnson, and D. Roberts. Wrappers to the Rescue. In *Proceedings ECOOP '98*, volume 1445 of *LNCS*, pages 396–417. Springer-Verlag, 1998.
[4] A. v. Deursen, L. Moonen, A. v. d. Bergh, and G. Kok. Refactoring test code. In M. Marchesi, editor, *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes (XP2001)*, pages 92–95. University of Cagliari, 2001.
[5] S. Ducasse. Evaluating message passing control techniques in Smalltalk. *Journal of Object-Oriented Programming (JOOP)*, 12(6):39–44, June 1999.
[6] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay. Back to the future: The story of Squeak, A practical Smalltalk written in itself. In *Proceedings OOPSLA '97*, pages 318–326. ACM Press, Nov. 1997.
[7] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, second edition, 1997.
[8] P. Schuh and S. Punke. ObjectMother, Easing Test Object Creation in XP, 2001. http://www.xpuniverse.com/2001/pdfs/Testing03.pdf.
[9] D. Thomas and A. Hunt. *Programming Ruby*. Addison Wesley, 2001.