

**Eclipse Summit Europe 2007**  
**TDD Symposium**

Position Paper by  
René Kießig ([r.kiessig@alea.de](mailto:r.kiessig@alea.de))  
21/09/2007

**ALEA GmbH**  
Leutragraben 1  
D-07743 Jena  
Tel. +49 (0) 3641 – 57 33 500  
Fax +49 (0) 3641 – 57 33 507  
[info@alea.de](mailto:info@alea.de)  
[www.alea.de](http://www.alea.de)

## **Companywide cumulative Test-Driven Development**

Writing this paper I tried to give an overview of our team's way to test-driven development.

We started with experience of afterwards written unit tests and the wish to write the tests first this time. Focusing on staying agile and continuously adding functionality we needed much time simple to come close to test-driven development.

Trying to take an objective view on our status quo we see there are still much problems to solve and things to improve. We hope others had as well these stones on their ways – we would like to listen to their stories and solutions.

### ***Background - our team***

We are a start-up company developing the three tier client server application Amc<sup>2</sup> for distant commerce companies. We use the Eclipse RCP and the c.a.r.u.s. cJEF framework. Our team started with 8 developers and increased after 10 months for the last one and half years to 16 developers now. We plan to deliver our software first time at the end of this year.

The initial team members and the management had experience with software projects of this kind. Focusing on testing: Manually processed test scripts and afterwards written JUnit Tests. They understood this was not the completely right way of assuring software quality.

As a result of their experiences for Amc<sup>2</sup> principles of Agile Programming should be implemented. The development should happen in iterations and test-driven.

Performing the development in iterations worked quite fine. We ended up with iterations of 4 weeks developing and 2 weeks testing. This helped us to stay in the plan for almost two and a half years now.

Developing test-driven did not work so fine. In the beginning only the basic life cycle of models was tested by using the cJEF frameworks mechanisms and JUnit. Mostly this tests were written

afterwards and did not help a lot. So less tests were written. Actually only one team member left developing test-driven.

### ***How to set the ball rolling?***

The situation changed when a new colleague joined the company. He brought along good experience with test-driven development with JUnit. When creating a new module with another team member they did this test-driven and by programming as a pair. The implementation of this module was quite successful.

The usual developing of the application went on without tests written. After this part project the new colleague kept the only one – beside the other colleague – producing automatic tests when implementing another new module alone. The next module which was developed test-driven was again done by a pair of a new colleague and another third team member.

All three test-driven implemented modules were quite independent. How to transfer this to rest of the application? In our team there seemed to be a critical mass. The two new team members and the initial team member motivated each other, exchanged experiences how to test in the cJEF frameworks environment and improved the available test mechanisms by creating an own TestCase class.

The spreading of test-driven development to other team members happened by giving workshops on how actually to write Unit tests, developing in pairs with other programmers and continuous impulses from the management. The joining of a QA specialist to our team was important, too.

The time the developers spent for testing manually in the end of an iteration got less. Instead they fix bugs, write more tests and documentation or concentrate on no impact tasks now.

### ***The Situation now***

At the moment we execute about 2400 JUnit tests in 400 test classes in 80 suite classes with CruiseControl every night. The result – committed changes, log outputs and the detailed nicely formatted test statistics – is accessible for every developer.

CruiseControl runs – after getting all changes, cleaning and updating the database – all JUnit tests and suites which follow the name convention.

During the day after every block of commits to the repository a special test suite is executed. This suite contains a selection of fast running tests covering the applications main functionality and aims to complete in half an hour.

Tasks are defined for programmers for failed tests and FindBugs errors found in the CruiseControl runs.

For new models and new functionality we aim to write JUnit tests before. Bugs should be reproduced by tests in advance to fixing them.

The tests we organize as common in a hierarchy of test suites.

### ***How we write our tests***

Usually we focus first on tests that prove the correct functionality – we call them positive tests. Afterwards with lower priority we create relying on the positive tests negative ones expecting exceptions and so on.

When adding a new business logic class we made good experience with: First writing and failing tests to instantiate and persist it and to remove the instance (we call them life cycle tests). Then modelling and generating the class and because of still failing tests creating the database table. Afterwards we create on top tests and code for constraints and functionality.

For the implementation of new business logic requirements we like to test it in a top down manner: We write a test for the simplest example. Then we add the code to satisfy the test. When realising other functionality (methods, classes etc.) must be implemented before, we switch to writing the tests and code for this functionality. If there are again must be implemented something before, we switch again.

We test the business logic classes within their context. We start up a complete application instance. We create many business logic class instances around the class to test. To easily create these models we use the cJEF frameworks mechanisms: When referenced first they are created and persisted. When modified by a test they are removed so they are created again clean when referenced next time in another test. We do not use special mock objects.

Opening the schoolbook we understand, we actually do not write unit tests but a kind of integration tests with JUnit. We explain it with the strong coupling with the cJEF framework. We saw no reasons why this should be a problem. But maybe it is one reason for the problems we have.

### ***Problems to solve***

Regularly some errors paralyse completely the nightly CruiseControl run. This can be a complete failed build or a decrease of the success rate to about 10%. The still succeeding tests belong to code which is quite independent.

Having theses identified errors or failures – low or high impact – we face normally difficulties to find their reasons – where in the code and whom of us. We thought about retesting the failing tests increasingly for every revision committed during the day.

The regularly execution of the fast running suite during the day should be a solution. Of course it shows paralysing errors earlier by being paralysed before and helps us a bit to find the reason. But it does not avoid that often enough the nightly run is not meaningful.

The second aim of the fast running suite is that every developer runs it before he commits changes which could have a bigger impact. Being able to cover the the main functionality it needs an hour to pass – too long everyone would execute it as often as he should.

Often the instantiation of the required business logic class instances hierarchy fails. Regularly the removing of these instances does not work because of database constraint violations. Continuously it costs manpower to organize the correct order of how the instances are removed.

We tried but could yet not reach the 100% success rate of the nightly test run. We needed almost half a year to increase it from about 30% to 80%.

Every developer of our team – one less, one more – has problems to discipline himself to continuously develop test-driven. Often enough tests are written afterwards or forgotten.

### ***Further things to achieve, questions to answer***

Tests made us think about our code, about our architecture, about our team organization and the way we test:

- We need to improve the test model infrastructure, especially removing them. Maybe there is another way of providing a clean test environment for each test again.
- We have to better organize our tests: Which tests or suites to execute in the night and during the day? How important or basic are they? Who is responsible for maintaining and fixing tests?
- How to increase the code coverage and test quality? When to stop writing tests? How to observe the coverage? How to useful integrate tools like EclEmma, Clover?
- How to decrease the complexity of tests and their dependency? How to write tests to immediately identify the first failing test and the after-effects? Can tools help us to find the first failing test or even the code (maybe in combination with coverage analyses)?
- What about separation unit and integration testing? Can we benefit from Fit/Fitness, how can we integrate it with CruiseControl?
- The complexity and dependency of the tests shows the the complexity and dependency of the code. Which strategies or tools could we use to solve this problem?
- Is the approach of positive and negative tests a good one? Are there better ways to let the tests grow?
- Are there further principles and experiences of agile programming concerning testing?

For these points we look for solutions or alternatives. For some we need only to give it a try, to integrate it, to use it companywide.

## ***What we understood***

- Important is the continuous push of the management toward test-driven development.
- Pair Programming increases the quality of the code, as well as the quantity and quality of the tests.
- New positive impacts come by new colleagues.
- It is very difficult to switch to test-driven development for already existing modules while still producing new functionality. This is true also for using better/other approaches.
- Only with a continuously running daily test run we have an overview over our tests and their success rate – without we can not value our tests.