

Eclipse Summit Europe 2007 – TDD Symposium

Getting Test Driven Developers – A Consultant’s Notebook

Position Paper by Marc R. Hoffmann (hoffmann@mountainminds.com), 22/08/2007

As a software consultant you may sometimes have the chance to launch a new project where its management is aware of the modern software development processes from the beginning, or on the other hand (and more likely), fixing a project that is already in serious trouble. In any case, test driven development will come into picture and it might be on you to establish a TDD processes. As with any methodology the primary obstacle is people accepting and adopting new ideas and throwing away the established but outdated processes.

So how do you get together the critical mass for a test driven developer team? How can you successfully get around “We always did it that way”¹?

This position paper outlines some important factors we identified in consulting projects over the last years while working with development teams on TDD techniques.²

Factor 1: Tools

First of all, test driven development requires a **basic tool kit** for easily creating and executing test cases.

- Select a simple **unit test framework** and make sure all developers have access to all required libraries and utilities.
- Provide **ready-to use setups** with **documentation** and **working examples**. Do not expect your team members to figure out on their own how to put everything together. This will help make sure everyone is on the same “page”.

Most of the time plain JUnit (or comparable frameworks for other languages) will do the job. Eventually some additions for DB testing or mock-up object might be required. Keep it simple and try not creating fancy or complex test frameworks that in the end will limit the ability to execute the test cases in different environments.

A **test runner** is as important to your test cases as a compiler is to your code. Test cases that are not executed frequently are as useless as plain source code. The test runners used in your project should

- allow running test cases fast and frequently,
- be available for every team member and
- provide instant and useful feedback.

¹ Alternatively: „We never did it that way“

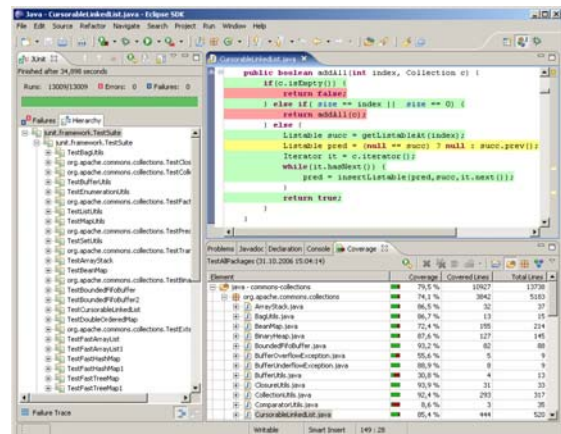
² While the ideas probably work for different technologies, the examples are primarily taken from practical experiences in Java, JUnit and Eclipse based projects.

Modern IDEs like Eclipse have great JUnit test runners integrated. Make sure your project layouts allow direct execution of all test cases within the IDE without additional local setups. Identify or invest in forming IDE/test framework experts in your team.

The quality of the tests itself can be determined with **code coverage** tools. Code coverage is not about telling the management that we now reached 82.7%. A coverage tool must be available to each developer during implementation and test writing. This allows identifying untested or dead code before it becomes part of an anonymous 17.3% figure. Tools like Clover or EclEmma bring this powerful approach directly into the IDE.

Beside the test execution by the individual developers, a continuous integration system should be in place to ensure the complete execution of your test suites in a defined environment and have the project status documented over time.

When implementing a CI server, work on the **fastest possible feedback cycle**, right from the code check-in to email notification about the build result. Timing is quite critical especially in globally distributed teams.



[1] EclEmma code coverage plug-in for Eclipse Instant feedback for every developer while writing test cases

Factor 2: Training

Every team member needs to know the basic ideas behind TDD and the deployed tools. If you browse the test code of an existing project you will always find some astonishing patterns. Don't hesitate to organize basic trainings, like for JUnit usage:

- Assertion mechanisms (avoid `if (value!=3) fail();`).
- Handling of unexpected exceptions (don't do it)
- Avoid "spaghetti" test cases
- Public `test*` () methods are just one way to declare unit tests.

Another obvious but crucial coherence is that **unit testing actually requires units**. The team should be aware of the approaches and adequate investment should be made in acquiring the related skills through training and constant coaching. Focus on techniques like:

- Decoupling through abstract interfaces
- Use dependency injection wherever possible

If a developer fails to write unit tests for a particular piece of code due to its current construction he might revisit his design. Therefore practicing unit testing will in turn create units, leading to a better code structure.

Good training measures require a thorough review of the existing code base and processes.

Factor 3: Rules

The team must work out basic rules to enable TDD within the project. Some guidelines:

- Write test cases together with your code, not later
- Run tests locally before you commit
- Commit frequently
- Don't commit broken code, fix it immediately
- Clean up compiler/checkstyle warnings
- Immediately investigate in failed test cases

Every project member should be aware of the project's quality and for example attentively observe the continuous build results.

Why not establishing a little game for this? Let's pass a moneybox around for charity. A team member who brakes one of the rules – e.g. by committing code that does not compile – makes a small contribution to the moneybox and becomes the new referee.³



[2] “Buildkuh” moneybox
Employed 01/2007 for a software project in banking industry

References

[1] Test Early, Employees of Stelligent Incorporated
<http://www.testearly.com>

[2] Continuous Integration, Paul M. Duvall et al, 2007

[3] EclEmma – Java Code Coverage for Eclipse
<http://www.eclemma.org>

About the Author

Marc works as an independent consultant and was involved in the development of several commercial Eclipse/RCP based projects and software products since 2004. For the last two years his professional focus was on establishing TDD environments and coaching development teams. Marc is the author of the EclEmma code coverage plug-in which was selected as the finalist in this year's Eclipse Community Award.

Contact

Marc R. Hoffmann, hoffmann@mountainminds.com
Mobile +49-170-3442727

Mountainminds GmbH & Co. KG
Nussbaumstr. 4
80336 Muenchen
Germany

³ I heard of teams doing similar games with liquors – not quite sure how this impacts software quality.