

ATDD in a Nutshell: Deliver the right thing

Ralf Wirdemann, November 2010, ralf@ralfwirdemann.de

Acceptance Test-driven Development (ATDD) is the practice of describing the functional requirements of a user story as concrete and automated examples prior the development of the story itself. ATDD facilitates communication and helps to develop software the customer really wants. It drives the development of a user story along its acceptance criteria. These are a set of business rules which need to be fulfilled in order to get the story done. Lets take the very common example of a bank account transfer:

As a customer I want to transfer money from my account to a destination account.

Lets drive out some simple acceptance criteria for this story:

- Balance will be reduced by the amount of transferred money
- Transfer must be legitimated by a valid TAN
- Customer must be logged in

While the title of the story is relative vague, its acceptance criteria make the story much more concrete by defining its core business rules. Acceptance criteria are the fixed corner stones of the story expressed in terms of the domain language. Acceptance criteria should be part of every user story in order to make the story better understandable, more binding and to provide measurable aspects to determine what's included and when the story is done.

Writing user stories and acceptance criteria is a good thing, but ATDD takes the "deliver the right thing approach" much further. In addition to writing acceptance criteria ATDD requires the product owner to explain each criteria in terms of real world examples. These examples become the acceptance tests for the explained criteria. Lets try this for the first criteria of the bank transfer story:

When I have an account with 100 EURO and transfer 20 EURO to another account, then my account balance should be reduced to 80 EURO.

This test is a perfectly fine example for the first business rule of the user story: The balance will be reduced. Every customer or developer who reads this test understands what the story does. And this is one of the core values of ATDD: Write examples for your business rules in a way the team understands your story. Acceptance criteria and associated tests expressed as real world examples are great tools to get a common understanding of the requirements. Customers and team need to talk to each other and build a common domain language and this talking is forced by writing examples as tests.

You can even take ATDD one step further by automating the acceptance tests which have been worked out. An automated acceptance test runs per click against the system under development without any further user interaction. The test checks the associated business rule and runs green if everything works as expected. This approach is quite similar to TDD: Write a failing (acceptance) test, write just enough (business) functionality to let the test pass.

Let's take the account reduction example from above and try to write a simple acceptance test for it:

```
public void shouldReduceSourceAccount() {
    givenAnAccountWith100Euro();
    whenCustomerTransfers20Euro();
```

```

        thenSourceAccountBalanceShouldBe80Euro();
    }

    private void givenAnAccountWith100Euro() {
        this.customer = BigBank.login("ralf", "sword");
        this.customer.account.setBalance(100);
        BigBank.updateAccount(account);
    }

    private void whenCustomerTransfers20Euro() {
        BigBank.transferMoney(this.customer.account, destAccount, 20);
    }

    private void thenSourceAccountBalanceShouldBe80Euro() {
        Assert.assertEquals(80, this.customer.account.getBalance());
    }

```

The acceptance test is written in terms of the well known ATDD terminology given/when/then. The "given" part formulates the precondition of the test. The "when" part describes the action triggering the business rule under test. And the "then" part describes the expected outcome. In the example above that is the account balance which should be reduced to 80 Euro after the transfer function was triggered in the given-statement. The interesting part of the test is the `transferMoney` method called on the system under development. If you implement the test without providing the functionality behind `transferMoney` the test will fail. In order to get it passed you have to provide just enough functionality to reduce the source account to 80 Euro. And this is what automated ATDD is all about: TDD on business level.

Acceptance criteria and their associated tests measure the progress of the story development. If you visualise the not yet implemented criteria or if you provide your product owner with a button to run your automated tests he will always be able to see where the story currently stands. The acceptance criteria drive the development progress. There will be no new code without a failing test. The story is only done when all criteria are described, automated by tests and all tests succeed. If you follow this approach you will not only drive out your functionality by tests. You will also build a continuously growing network of regressions tests which constantly validate the business rules of your system. New functionality breaking existing business functionality will be reported immediately by a failing test which prevents you from delivering the software.

But ATDD is more than a testing tool. It is much more a communication and cooperation tool. ATDD requires the product owner to provide acceptance criteria for all of his user stories. In order to explain these criteria he needs to explain the criteria to his team in terms of real world examples. Therefore criteria forces a lot communication prior the real story development and communication is the key factor to deliver the right thing: Software the customer really wants.