JAOO conference 2007
September 23 - 28, Aarhus, Denmark

Test-Driven Development

Erik Doernenburg & Martin Fowler
ThoughtWorks

Agenda

Iteration 1
Test-driven development

Iteration 2
Interaction-based testing

Iteration 3
Lightweight containers

The XP values

Simplicity

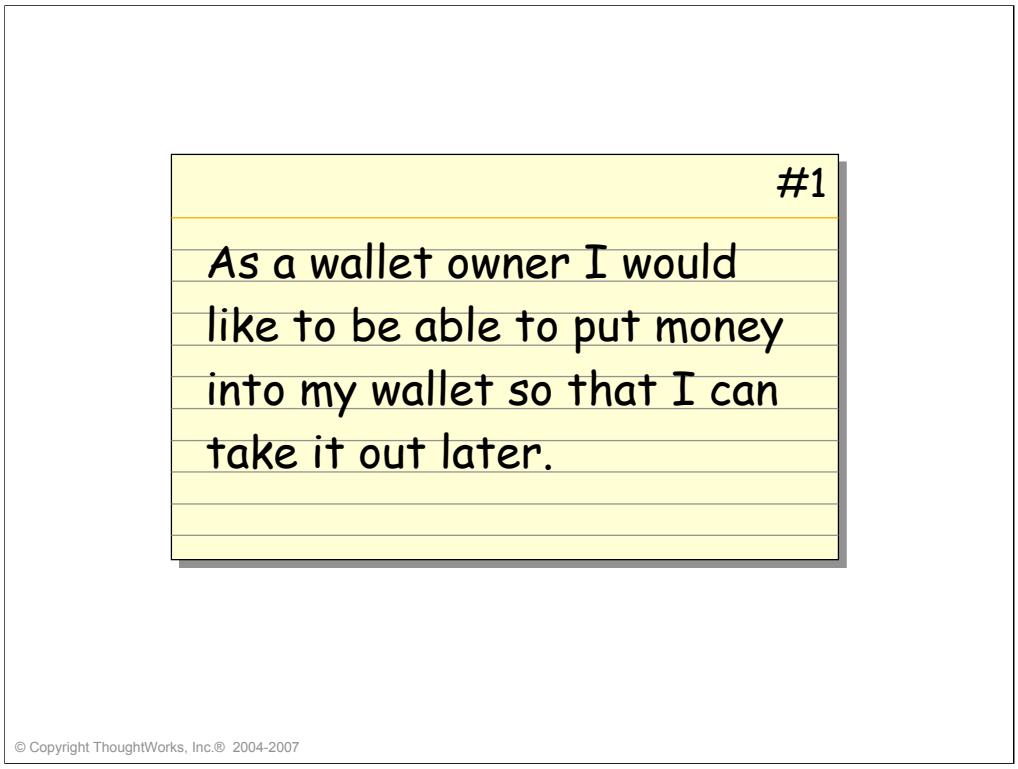
Communication

Feedback

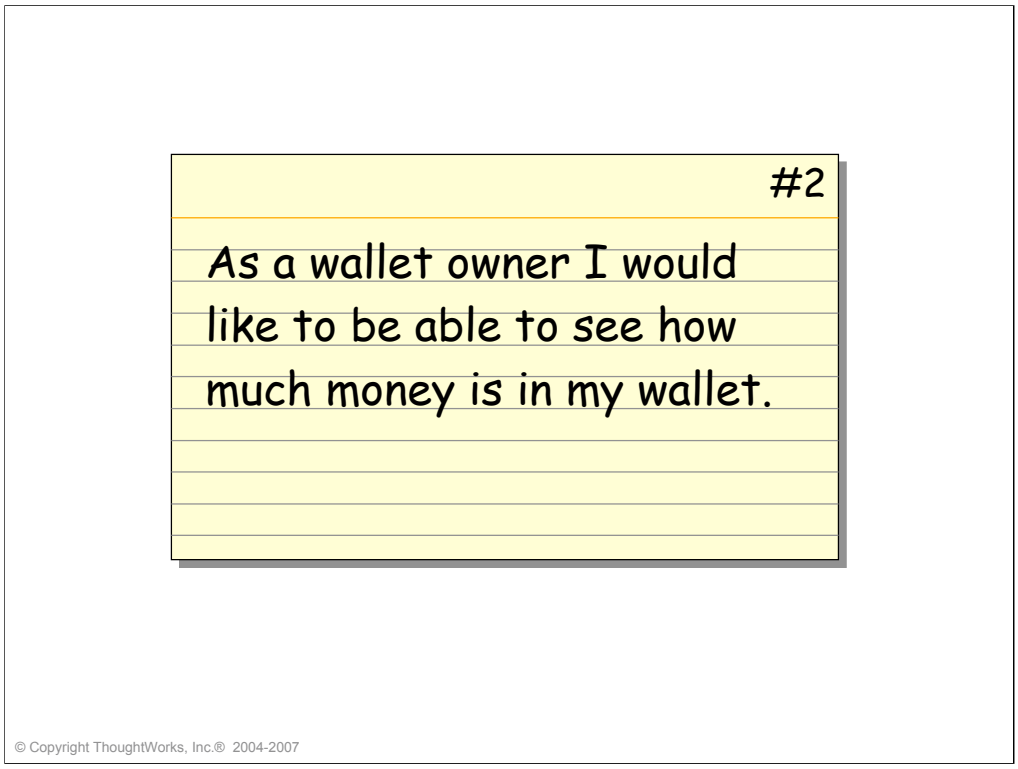
Courage

Respect

Iteration 1
Test-driven development

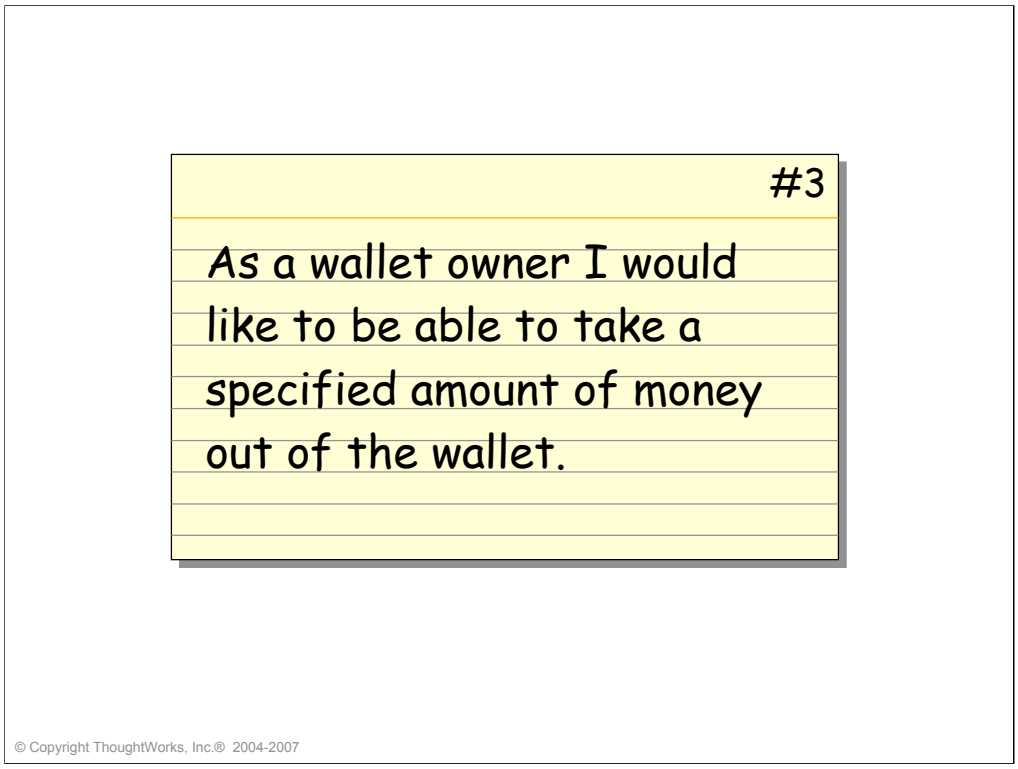


- Use long test method names. Show agiledox.sf.net
- Use odd numbers/values, i.e. don't use 10 all the time, use 5945 and 21423432. This might expose unintended dependencies.
- Can make the test pass by returning a hard-coded value. Write two tests to force real implementation. (Discussion: This is equivalent to giving two points in the problem space, rather than one. How much does that narrow the specification? Kent says: "Use three!")
- Can you take the money out twice? Was this specified?



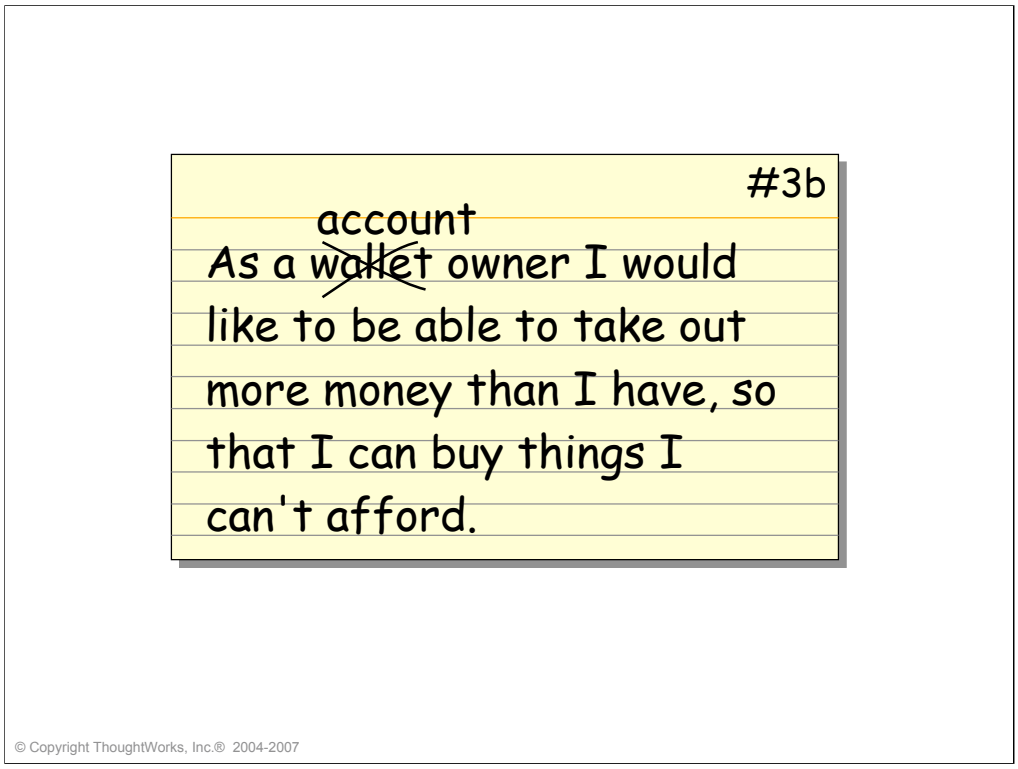
•Easiest thing to make the test pass: Implement viewMoney in terms of takeOutMoney. Write a test to prove that viewMoney doesn't have side effects.

•Discussion: how much do we need to specify things that shouldn't happen? Aren't tests just a specification of what should happen?



•Refactor, i.e. make takeOutMoney() use new takeOutMoney(long amount).

•Are the business users happy? No, you can take out more than is in the wallet! Missing requirements…

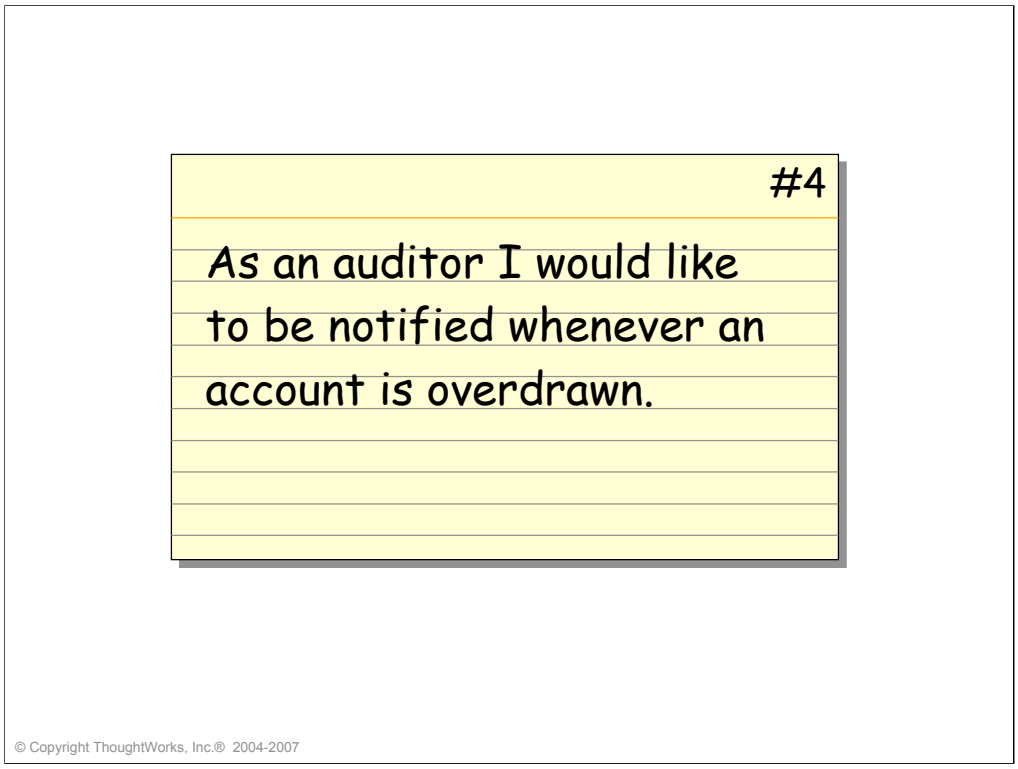


•People change their mind… Requirements always change. Embrace change!

•Rename classes to reflect current usage, wallet becomes account (cf. Eric Evan's ubiquitous language). Maybe stop short of renaming all variables in the test cases. Note that many IDEs can do this for us!

•Delete unneeded code, e.g. NoMoreMoneyException. Do not second-guess that it may be needed again in the future.

Iteration 2
Interaction-based testing



•Discuss the idea of an “active” domain model, but introduce service layer, i.e. AccountService

•Task 1: Test the basics first, i.e. does withdraw actually work (use amount that leaves negative balance)

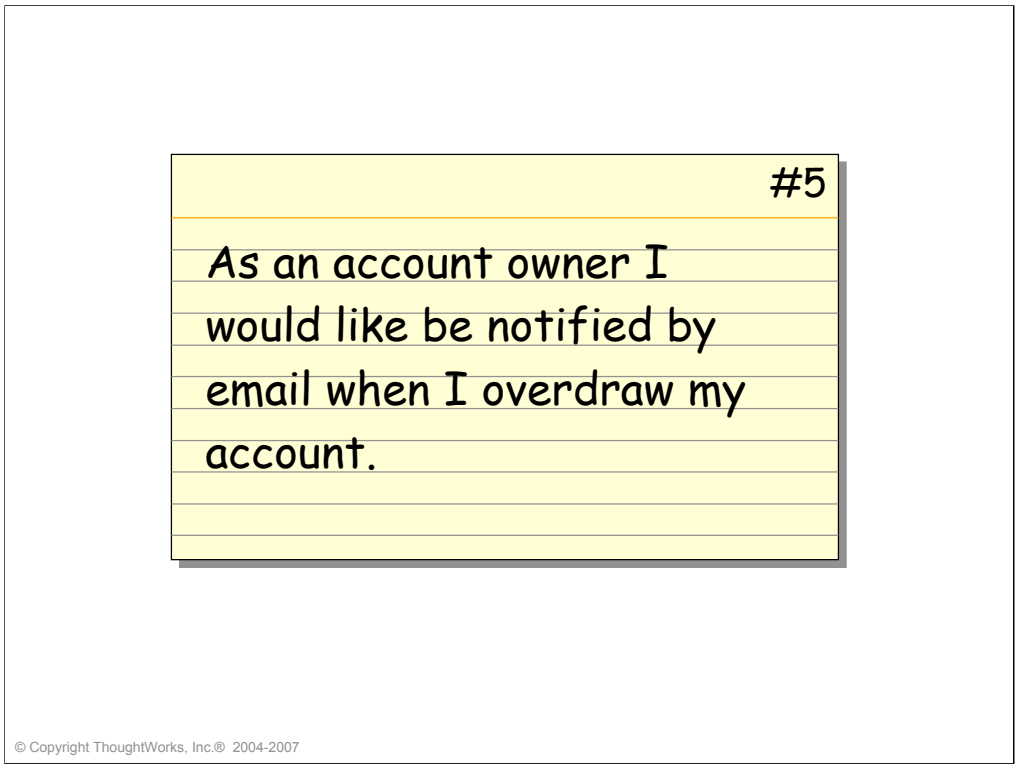
- •Test with real object, then show how it’s done with a mock
  - •Just put in mock, without expectation – fail fast!
  - •Show how the mock test fails when the takeMoney() method call is taken out. – To verify or not to verify?
- •Sometimes domain objects work just fine…

•Task 2: Now add a test for the audit functionality

- •Don't get distracted by infrastructure. Create an interface for MonitorService, discover its methods, use a Mock for the interface.
- •Don't forget to create the isOverdrawn() method on domain object (separation of concerns)
- •Don't retest final balance

•Run all the tests again

- •Use null object pattern when we add default constructors to make first tests work; do not check for null MonitorService.



- Now add a test for the email functionality
  - Create an interface for MailService, use a Mock for the interface.
  - Baby steps, just use a hard-coded subject first
- We need a bit more, i.e. the recipient address
  - Put email address directly into account
  - Show the difference the 'change signature' refactoring makes
- Introduce 'Message' parameter object; should be immutable
  - Show hasProperty() and and() jMock constraints – how do failures look, e.g. when looking for 'joe@bloggs.org'
  - Show custom constraint that uses string equals() methods
  - Show custom constraint with assertEquals()
  - Show EmailServiceStub approach
- None of the above solutions is really good. What's the problem? – The 'new Message()' in AccountService
  - Why not just create the Message in test? – would need equals() method; which might be okay for a value object…
  - Or we just create a MessageFactory (cf. DDD)

#6

As an auditor I would like to be notified whenever more than £10,000 are withdrawn from an account in one transaction.

- [Skip this if time's short]
- Add logic to withdraw method in AccountService
- There could be some overlap in the test methods (what happens when we withdraw more than the limit in the mail test?)
    - Extract methods – but how to test, they are private?
        - Make protected, if tests in same package, or public if not
        - Use subclass
        - The public do(), private step1(), private step2() pattern…
    - Sometimes it's okay not to test the wrapper methods; because we have acceptance tests, right?

Verification Trade-Offs

Classic

- Testing independent of implementation
  - Design
  - Refactoring
- Mini-integration tests
- Inside Out

Mockist

- Avoids complex fixtures
- Encourages decoupling
- Fault has one Failure
- Outside In

Two Dichotomies

State Verification vs Behavior Verification

http://xunitpatterns.com

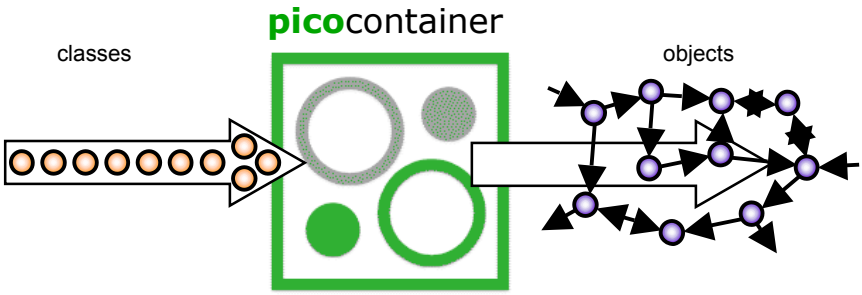
Classic Unit Testing vs Mockist Unit Testing

http://martinfowler.com/articles/mocksArentStubs.html

Iteration 3
Lightweight containers

•[Show next two slides first]
•Build a main method with dummy services that write to System.out
•Show how wiring can be done by DefaultMutablePicoContainer
•Show how the same thing would be done with Spring in an XML configuration file
•Show how the tests become awkward when we replace DI with SL in the services

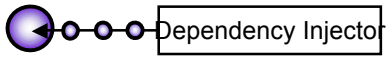
A Lightweight Container



Dependency Resolution

- Service Locator

- Dependency Injection



Odds and Ends
Object Mother

- How can we reduce duplicated code in tests?
  - Add abstractions – but do we want to debug them?
  - What is the duplication?
    - Use an ObjectMother to set up domain model