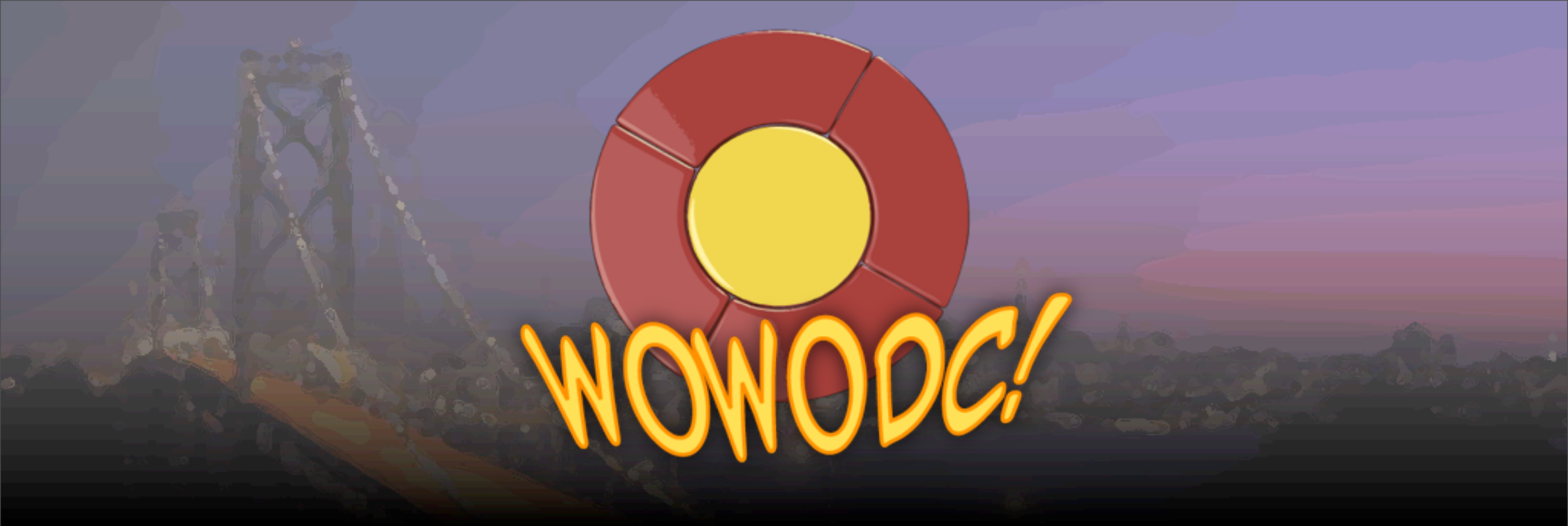


Test Driven Development with WebObjects

Denis Frolov
Demax

What Is TDD?

"Never write new functionality without a failing test" - this phrase is a golden rule of TDD and pretty much describes the core idea behind it. Test-driven development is a software development technique of writing tests before writing code.

TDD turns testing into a design activity. We use tests to clarify our ideas about what we want the code to do. With TDD we think about the tests as a client of the code before diving into the implementation details.

And if we write tests all the way through the development process, we can build up a safety net of automated regression tests that give us the confidence to make changes.

So, basically TDD is about two things - building up a safety net of automated tests and driving the design of your project.

The Process

Monday, June 15, 2009

TDD process consists of two loops - outer and inner. When we're implementing a feature, we start by writing an acceptance test, which exercises the functionality we want to build. While it's failing, an acceptance test demonstrates that the system does not yet implement that feature; when it passes, we're done. Underneath the acceptance test, we follow the unit level test/implement/refactor cycle to develop the feature.

At Demax we use Selenium browser testing framework for our acceptance testing level and JUnit unit testing framework for unit testing level. For some features unit level testing will not be needed - this can happen if your feature is all about integration of already existing modules.

Demo

http://code.google.com/p/website-monitor-demo/

Monday, June 15, 2009

To give you a feel of the workflow we use in our own projects, I'll demo a creation of a project from scratch using WebObjects, Wonder, D2W, Selenium, JUnit, and Mockito.

* New Wonder Application
* TODO.txt
* ERSelenium dependency
* ListWebsites.sel
* Run app. Change WOPort in Debug Configuration to 55555.
* Run test. Fail - need "Selenium" and "cleanup". Create class and method
* Run test. Fail - need "createExampleWebsite". Create method. Discover need of Website entity
* WebsiteBuilder.build() implementation
* Model, entity, eogen, migration
* DB connection properties
* Migration properties
* Add mysql-connector-java-5.1.7-bin library
* Run test. Pass example object creation. Fail on websites list. Need D2W.
* D2W Frameworks: ERDirectToWeb, JavaEOProject, JavaDirectToWeb, JavaDTWGeneration
* Factory.listWebsites
* Run test. Fail - need PageWrapper. Add PageWrapper.
* Run test. Fail - need Edit/Check links.
* Add user.d2wmodel. Copy/paste rules.
* Add WebsiteController with "edit" and "check" method stubs.
* Run test. Pass. Need more simple layout with custom templates. Meet DMSimpleLook.
* DMSimpleLook overview - simplistic templates with clean css layout using yui css foundation. Can be shared by different projects.
* Add DMSimpleLook dependency.
* ognl.parseStandardTags=true
* Add css links
* Wrap contents into yui layout div ( )
* Add base.css to project
* Run test to make sure we didn't break things.
* CheckAvailableSite.sel
* "listWithExampleWebsite" setup method instead of "createExampleWebsite" + "open /" combination.

Top 5 Benefits

So far, we've implemented 2 of the 6 user stories. Unfortunately, my time is running low, so I'll have to leave the implementation of the remaining features as an exercise for the reader and will outline the top 10 benefits we gained by applying TDD to this simple project.

Freedom & Courage

All written code is covered by tests. This gives us a great level of trust in the code. We now have enough courage for merciless refactorings, we are no longer afraid of running with a trunk version of Wonder, we can push out new features to production on Friday nights and sleep well all night long.

2

Shorter Implementation Time

While it is true that more code is required with TDD than without TDD because of the unit test code, total code implementation time is typically shorter. Large numbers of tests help to limit the number of defects in the code. The early and frequent nature of the tests helps to catch defects early in the development cycle, preventing them from becoming endemic and expensive problems. Eliminating defects early in the process usually avoids lengthy and tedious debugging later in the project.

3

Modularized, Flexible, and Extensible Code

Monday, June 15, 2009

TDD can lead to more modularized, flexible, and extensible code. This effect often comes about because the methodology requires that the developers think of the software in terms of small units that can be written and tested independently and integrated together later. This leads to smaller, more focused classes, looser coupling, and cleaner interfaces. The use of the Mock Object design pattern also contributes to the overall modularization of the code because this pattern requires that the code be written so that modules can be switched easily between mock versions for unit testing or "real" version for deployment.

4

Executable Description of What the Code Does

Monday, June 15, 2009

Indeed, if we look at Selenium and Unit test we'll quickly get an idea of the requirements. We can also use Selenium tests to quickly execute specific use cases which can involve complicated setup. This can save a whole bunch of time during maintenance.

5

No "Gold Plating"

Having a clear acceptance criteria of when to stop lets us know when we've done enough. This greatly discourages adding unneeded enhancements to a feature that is already finished.

TDD Can Be Hard

Monday, June 15, 2009

It will take time to get used to and can be painful especially if you're doing it in a legacy code base. Also, most probably you'll feel that you loose productivity at start.

Books

- “Test Driven Development By Example” by Kent Beck
- “Growing object-oriented software, guided by tests” by Steve Freeman & Nat Pryce http://mockobjects.com/book/index.html

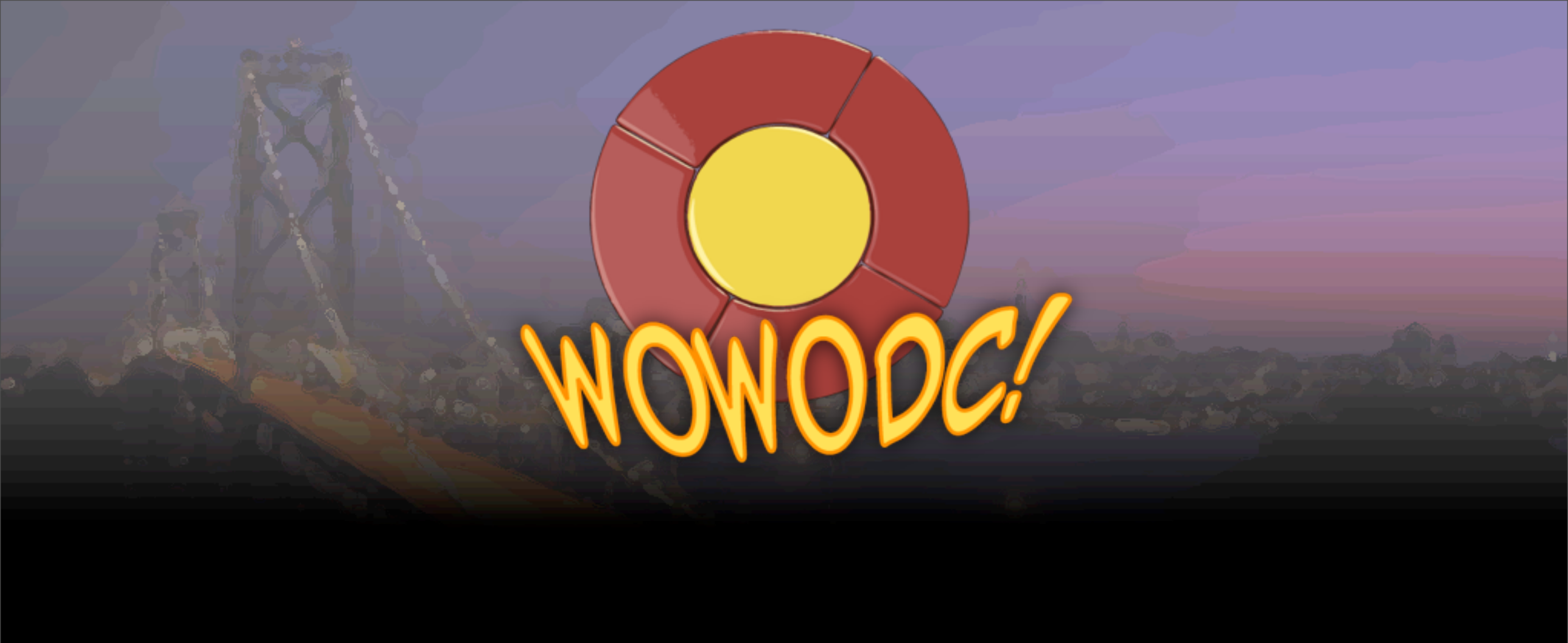
Monday, June 15, 2009

"Test Driven Development By Example" by Kent Beck is an obvious choice. Another option is http://mockobjects.com/book/index.html.

Acceptance Tests Are Your Best Friends

Monday, June 15, 2009

Even if you don't dive directly in TDD - you should definitely start testing whatever you can. Start with acceptance (Selenium) tests - they are much easier to write and work great with legacy code base. If I had to choose between acceptance tests and unit tests - I would definitely choose acceptance ones.



Q&A

Behind the scenes setup:
* WOLips - EOGenerator: WonderEntity.java
* Disabled serialVersionUID warns
* Existing database SiteMonitor
* Problems view - Configure Contents - Scope - On any element in same project
* WOPort = 55555