

Behaviour Driven Development with Instinct

Workingmouse Tech Talk
February 2007

Tom Adams

Overview

- Part 1 - Introduction to Behaviour Driven Development
- Part 2 - Instinct overview
- Part 3 - Instinct demo
- Part 4 - Miscellany

Part I - Introduction to Behaviour Driven Development

TDD adoption profile

- The developer starts writing unit tests around their code using a test framework like JUnit or NUnit.
- As the body of tests increases the developer begins to enjoy a strongly increased sense of confidence in their work.
- At some point the developer has the insight (or are shown) that writing the tests before writing the code, helps them focus on only writing the code that they need.
- The developer also notices that when they return to some code that they haven't seen for a while, the tests serve to document how the code works.
- A point of revelation occurs when the developer realises that writing tests in this way helps them to “discover” the API to their code. TDD has now become a design process.
- Expertise in TDD begins to dawn at the point where the developer realizes that TDD is not about testing, it is about defining behaviour.
- Behaviour is about the interactions between components of the system and so the use of mocking is a fundamental to advanced TDD.

<http://behaviour-driven.org/TDDAdoptionProfile>

Testing Maturity Phases

- Phase 1: Mostly state based testing – not using mocks or stubs
- Phase 2: Mostly state based testing – with using mocks or stubs only for database access in the layers above it
- Phase 3: Shifting from state based testing to more interactive style of testing
- Phase 4: Accent on the interactive style of testing. Some initial steps towards Behavior Driven Design
- Phase 5: Behavior Driven Design with very careful consideration on Interactive based testing

http://igorstoyanov.blogspot.com/2005/10/stubs-or-mocks-state-or-behavior_30.html

Introduction to BDD

“It’s about figuring out what you are trying to do before you run off half-cocked to try to do it. You write a specification that nails down a small aspect of behaviour in a concise, unambiguous, and executable form. It’s that simple. Does that mean you write tests? No. It means you write specifications of what your code will have to do. It means you specify the behaviour of your code ahead of time. But not far ahead of time. In fact, just before you write the code is best because that’s when you have as much information at hand as you will up to that point. Like well done TDD, you work in tiny increments... specifying one small aspect of behaviour at a time, then implementing it.”

Dave Astels

Problems with TDD

- Name implies testing is for testing only
- Gives rise to the view that testing is an optional extra, a nice to have
- “I’m not going to write all those tests”
- “It’s really simple code, it doesn’t need to be tested”
- “Testing is a waste of time”
- “I’ve done this (loop/data retrieval/functionality, etc) millions of times.”
- “We test after the code is done”
- “That’s what we have a testing person for”
- “We can’t spend that time now”

Core BDD concepts

- Behaviour context - Sets up a context in which certain behaviour is expected. Equivalent to a single test class (extends TestCase)
- Specification - Given a certain context, what should happen? Equivalent to a single test method (testIWantSomethingToHappen)
- Brian Marick calls BDD example driven development, and calls specifications “examples”
- Instinct has purposefully avoided weighing into this debate, and instead uses BDD nomenclature

What does BDD give me?

- Shift in vocabulary, BDD focuses on “getting the words right”
- Thin wrapper on top of TDD?
 - BDD is not a paradigm shift, it is an evolution of TDD based on experience
 - BDD is a pragmatic refactoring of TDD
- BDD is basically what TDDers have been doing for a while
- Specification structure is not coupled to production code structure
- BDD has a strong emphasis on mocking
- TDD “flow” stays the same - Test, code, refactor

Part 2 - Instinct Overview

So what is Instinct?

“Instinct is a Behaviour Driven Development (BDD) framework for Java. Inspired by RSpec, Instinct provides flexible annotation of contexts, specifications, mocks, etc. (via Java 1.5 annotations, marker interfaces or naming conventions); automatic creation of test fields (subjects, fixtures, dummies, mocks & stubs); a verification API; and JUnit test runner integration.”

What! Another testing framework?

- Instinct is BDD framework, so has a slightly different focus to conventional testing frameworks
- It formalises definitions (by including them in the syntax of the framework) of common test objects such as subjects, mocks & stubs
- It Includes test objects directly into the lifecycle of a specification
- It does away with needless infrastructure setup such as stub/mock creation
- Flexible marking of test objects - specifications, mocks, stubs, dummies, etc. based on annotations, marker interfaces and naming conventions

What! Another testing framework?

- Test objects are explicitly marked with their function
- Simplification of mocks and controllers, the mocking API only provides access to the mock, doing away with the need to access the controller and manage two objects
- It removes the need for concrete class inheritance (which most testing frameworks now support anyway)
- It makes use of Java 1.5 features in order to simplify testing, such as annotations and typesafe mock creation
- It embodies lots of common code usually created on TDD projects as Open Source Software making it available outside individual projects

What! Another testing framework?

- I wanted to explore the state vs. interaction testing debate
- Other Java BDD frameworks (such as jBehave) have a different focus (& I hadn't seen JDave yet!)
- Instinct is focused on unit/atomic (tight) testing, but like JUnit et al. can be used for integration/acceptance/etc.

Assumptions

- Value simplicity
- Simplicity may mean magic, so investment should be minimal
- The framework is everyone's friend - will reflectively invoke things if need be
- Test code is not like production code, the same rules don't apply

Core Instinct goals

- Flexibility
 - Annotations, marker interfaces, naming conventions
 - Should not need to extends base class for framework to work
- Explicitness
 - Mocks, stubs, fixtures, subjects are explicitly marked, simplifying
- Minimal impact
 - “Convention over configuration”

Core Instinct goals

- Java 1.5
 - Type safety - inferred return types
 - Static imports (syntactic sugar)
- Ease of adoption - JUnit runners, Ant, etc.
- Anti-bloat

Features

- Marking \Rightarrow annotations, naming conventions, marker interfaces, method signatures(?)
- Automatic test field creation \Rightarrow dummies, mocks, stubs, fixtures, subjects
- Mocking \Rightarrow jMock built with simplifications
- Verification API \Rightarrow Assertions, mocking(?)
- Integration \Rightarrow JUnit & Ant
- JUnit feature parity

What do I get?

- Support for running specifications marked using Java 1.5 annotations
- Marker annotations for specification lifecycle methods
- An Ant task, providing aggregation of contexts and result output (similar to the JUnit Ant task)
- An integrated mocking API, built on jMock 1.1
- A simple Verification class (will change)
- JUnit test suite
- A sample project showing how to use Instinct

Nomenclature

- Subject \Rightarrow The class whose behaviour is under scrutiny
- Test double \Rightarrow An implementation of an interface (or extension of a class) that is only used for testing
 - Dummy \Rightarrow Dummy objects are passed around but never actually used
 - Stub \Rightarrow Stubs respond to method calls made during a test by providing canned answers
 - Mock \Rightarrow Mocks are more advanced stubs, that respond to calls & verify expectations
- Fixture \Rightarrow A fixture is a known set of test data or helper

Part 3 - Instinct Demo

Part 4 - Miscellany

Auto test field creation

```
public final class DefaultSetupElementCreatorUnitTest extends PrimordialTestCase {
    private SetupElementCreator setupElementCreator;
    private BatchRunSubmission batchRunSubmission;
    private String setupName;
    private String setupDescription;
    private BatchRunPart[] batchRunParts;
    private ProcessingProfile processingProfile;
    private ProcessingProfileDataFetcher mockProcessingProfileDataFetcher;
    private UnbarcodedElementCreator mockUnbarcodedElementCreator;
    private RunSubmissionType mockRunSubmission;
    private SetupType mockSetup;
    private BatchRunPartFinder mockBatchRunPartFinder;

    protected void setUpFixtures() {
        setupElementCreator = new DefaultSetupElementCreator(mockProcessingProfileDataFetcher,
            mockUnbarcodedElementCreator, mockBatchRunPartFinder);
        batchRunSubmission.setProcessingProfile(processingProfile);
    }

    public void testProperties() {...}

    public void testCreateSetupElement() {
        BatchRunInstance batchRunInstance = batchRunSubmission.getBatchRunInstance();
        expectOneCallTo("addNewSetup", mockRunSubmission).will(returnValue(mockSetup));
        expectOneCallTo("setSetupName", mockSetup).with(eq(setupName));
        BatchRunPart firstBatchRunPart = batchRunParts[0];
        expectOneCallTo("findByBatchRunInstance", mockBatchRunPartFinder).with(same(batchRunInstance))
            .will(returnValue(batchRunParts));
        setupElementCreator.createSetupElement(mockRunSubmission, batchRunSubmission);
    }
}
```

Auto test field creation

```
public final class DefaultDeferredDeliveryHerderIntegrationTest extends WebServiceWiredTestCase {
    private HibernateOperations wiredHibernate;
    private TableCleaner wiredTableCleaner;
    private GenericFinder wiredGenericFinder;
    private DeferredDeliveryHerder deferredDeliveryHerder;
    private BatchRunInstance batchRunInstanceWithNoParts;
    private PersistentEntityReference[] persistentEntityReferences;
    private BatchRunInstanceFinder wiredBatchRunInstanceFinder;
    private int uniqueKey;
    private Uuid uuid;
    private DeferredDeliverableFinder mockEmailAndPrintFinder;
    private DeferredDeliverer mockDeliverer;
    private BatchRunSubmission mockBatchRunSubmission;

    public void setUpFixtures() {
        deferredDeliveryHerder = new DefaultDeferredDeliveryHerder(mockEmailAndPrintFinder, mockDeliverer, wiredGenericFinder)
        batchRunInstanceWithNoParts = wiredBatchRunInstanceFinder.findCurrentInstanceByNameAndApplication(
            BatchRunNameLookup.INTERNAL_AND_INSURE_BATCH_RUN_NAME_1, "INSURE");
    }

    public void testBatchRunInstanceWithNoPartsDoesNotDeliverRunSubmissionReference() {
        doNotCareExpectations(batchRunInstanceWithNoParts);
        createAndSaveBatchRunInstanceWithParts();
        deferredDeliveryHerder.herd(mockBatchRunSubmission);
    }

    public void testBatchRunInstanceWithOnePartDoesDeliverRunSubmissionReference() {
        BatchRunInstance batchRunInstanceWithParts = createAndSaveBatchRunInstanceWithParts();
        doNotCareExpectations(batchRunInstanceWithParts);
        expectDeliveryOfBatchRunSubmissionReference();
        deferredDeliveryHerder.herd(mockBatchRunSubmission);
    }
}
```

Auto test field creation

```
public final class DefaultDocumentFinderIntegrationTest extends WebserviceWiredTestCase {
    private DocumentFinder wiredFinder;
    private HibernateOperations wiredHibernateOperations;
    private TableCleaner wiredTableCleaner;

    public void testFindOneDocument() {
        TaskTrackingInfo tti = createTaskTrackingInfo();
        final Document[] inputDocuments = {createDocument(tti)};
        checkFindDocuments(tti.getTaskReference(), inputDocuments);
    }

    public void testFindTwoDocuments() {
        TaskTrackingInfo tti = createTaskTrackingInfo();
        final Document[] inputDocuments = {
            createDocument(tti),
            createDocument(tti),
        };
        checkFindDocuments(tti.getTaskReference(), inputDocuments);
    }

    public void testFindsNoDocuments() {
        TaskTrackingInfo tti = createTaskTrackingInfo();
        assertEquals(new Document[] {}, wiredFinder.findByTask(tti.getTaskReference()));
    }

    private TaskTrackingInfo createTaskTrackingInfo() {
        TaskTrackingInfo tti = (TaskTrackingInfo) getInstance(TaskTrackingInfo.class);
        wiredHibernateOperations.save(tti);
        return tti;
    }
}
```

Auto test field creation

```
public final class DefaultDynamicInterfacerIntegrationTest extends WiredTestCase {
    private IOException ioException = new IOException("xxx");
    private short uniqueShort;
    private DynamicInterfacer dynamicInterfacer;
    private InterfaceValidator wiredInterfaceValidator;
    private ProxyUtil wiredProxyUtil;

    protected void setUpFixtures() {
        dynamicInterfacer = new DefaultDynamicInterfacer(wiredInterfaceValidator,
            wiredProxyUtil);
    }

    protected String[] provideWiringFileNames() {
        return new String[]{"common.test.wiring.xml"};
    }

    public void testVoidMethod() {
        D d = new D();
        Runnable runnable = (Runnable) dynamicInterfacer.cast(Runnable.class, d);
        assertEquals(0, d.getRunCount());
        runnable.run();
        assertEquals(1, d.getRunCount());
        runnable.run();
        assertEquals(2, d.getRunCount());
    }
}
```

What's next?

- Improvements to the verification API (see Ben), possibly unify mocking & verification API
- Completion of auto test field creation
- Implementation of additional markers - naming conventions & marker interfaces
- Mocking improvements: ordering, type safety, jMock 2
- IntelliJ plugin
- JUnit XML test output formatters (?)

Differences

- No Success, failure and error; only success and failure, errors are considered failures
- Multiple marking schemes \Rightarrow annotation, naming convention, marker interfaces, others?
- Auto test field (double, fixture, etc.) creation

Other frameworks

- BDD ⇒ RSpec, JDave, JBehave, NSpec
- TDD ⇒ TestNG, JUnit, JTiger

Where to now?

- Get it! <http://code.google.com/p/instinct/>
- Talk about it! <http://groups.google.com/group/instinct-general>
- Learn! <http://code.google.com/p/instinct/wiki/Tutorials>
- Read the propaganda! <http://adams.id.au/blog/category/technology/instinct/>

References

- http://en.wikipedia.org/wiki/Behavior_driven_development
- <http://dannorth.net/introducing-bdd/>
- <http://blog.daveastels.com/articles/2005/07/05/a-new-look-at-test-driven-development>
- http://blog.daveastels.com/files/BDD_Intro.pdf