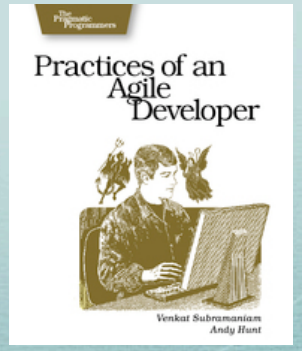
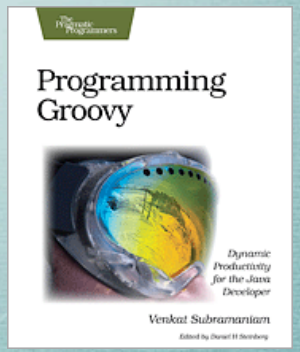
BDD IN JAVA AND GROOVY

```
speaker.identity {
  name        'Venkat Subramaniam'
  company     'Agile Developer, Inc.'
  credentials 'Programmer', 'Author', 'Trainer'
  blog        'http://agiledeveloper.com/blog'
  email       'venkats@agiledeveloper.com'
}
```



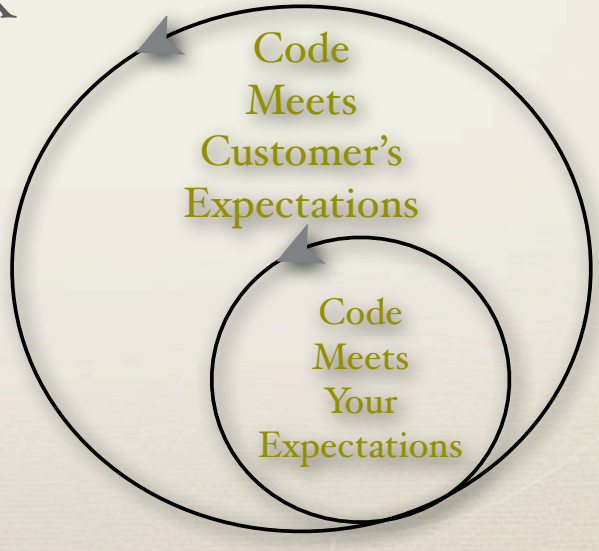


Abstract

- In this presentation we will take a look at what BDD is and look at tools to create them in Java and Groovy.
- What's BDD?
- Benefits of BDD
- Tools for BDD
- Creating BDD in Java
- Creating BDD in Groovy

Essence of Agility

- To create relevant working software
- Developing software is hard business
- How can you succeed?
- Feedback is essential
- Two kinds of feedback



Test Driven Development

- The word "Test" in TDD is a bit misleading
- It is not about verifying software
- It is an approach to developing software by way of writing code that exercises your code
- It helps you to
  - create a lightweight design
  - express behavior
  - create a form of highly expressive documentation
  - Keep an eye on code—to tell you if it begins to fall apart

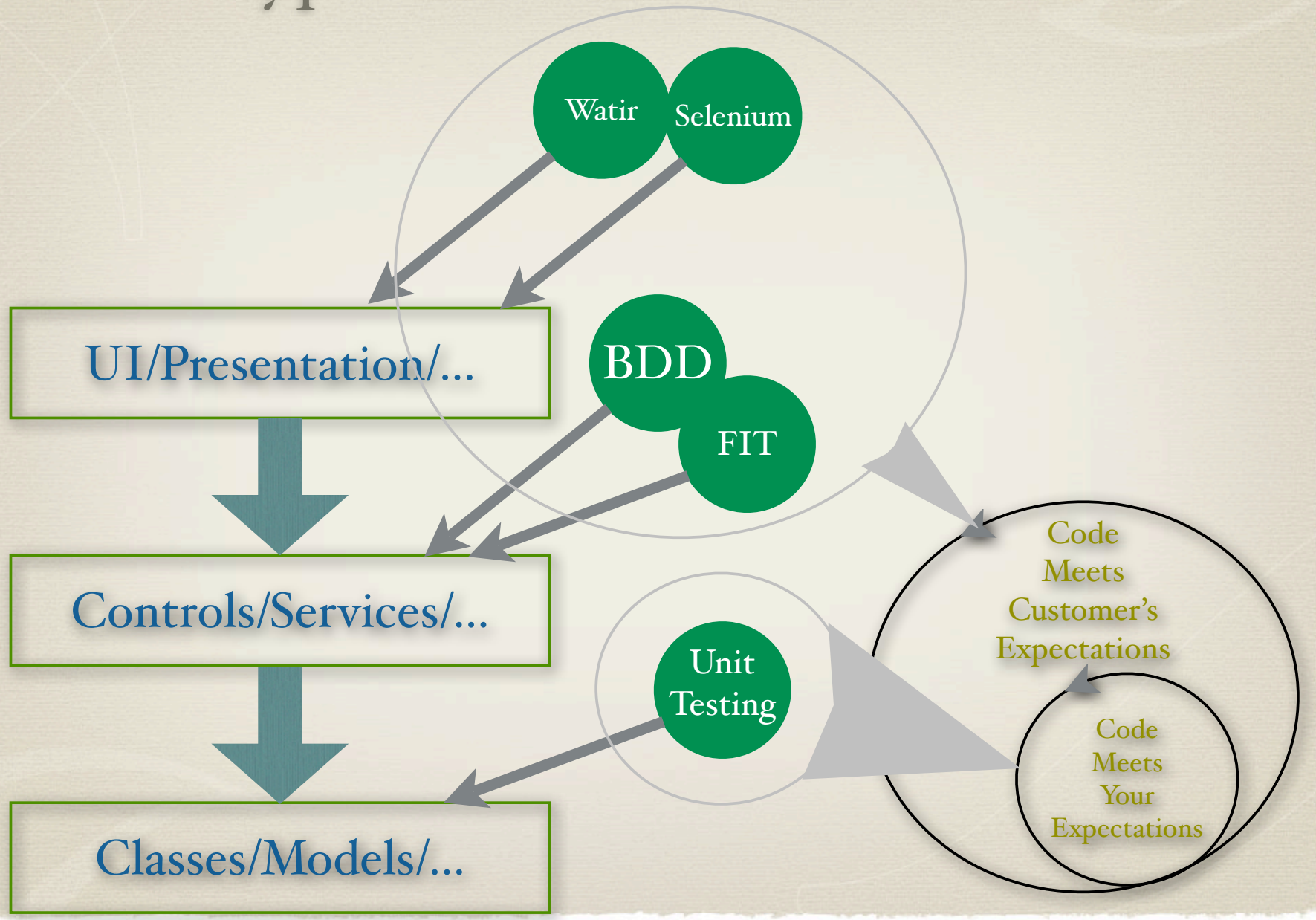
Unit Testing: Essential but not Sufficient

- Unit Testing is one (but not only) example of TDD
- Unit Testing tells us that code's meeting the programmer's expectations
- Very important to know the code continues to meet that expectation as software evolves to meet user's expectations
- But, how do you know what's the user's expectations?

Ways to express User's Expectations

- Use cases
- User Stories
- Agile projects tend to lean towards user stories
- Still, how to verify code continues to meet those expectations/requirements?

Types of Tests and Levels



Ubiquitous Language

- "A language structured around the domain model and used by all team members to connect all the activities of the team with the software"—Domain Driven Design by Eric Evans.

Executable Documentation

- We typically express requirements and system behavior in the form of documentation
- What if you can actually execute that documentation?
- You can show to yourself that the code is meeting the expectations
- Helps you to ascertain that code continues to meet those expectations
- Enter Behavior Driven Design
  - Introduced by Dan North

Behavior Driven Design

- It is a TDD approach
- It is a ubiquitous language
- It is an executable documentation
- It promotes communication
- Helps develop common vocabulary and metaphor
- Help you to get the "words" right
- Can be used by programmers, testers, business analysts, domain experts, and customers.

Behavior and Story

- You can use BDD to express Stories and Behaviors
- Story Framework and Spec Framework
- Stories correspond to User Stories—to express behavior at application level
- Spec or Behavior correspond to expectations at class level—to express behavior at service/component level
- These can help express requirements that can be specified, understood, and negotiated by developers, testers, business analysts, and business customers.

Behavior

- Each behavior is expressed as a test/*exercise* method
- It tells what the object *should* do
- Notice the keyword "should"—that's a main focus in BDD—the *should*s and the *shouldn't*s

Building Stories

- You may define user stories as a series of acceptance criteria as scenarios

- It has the givens, events, and outcomes
- That is
  - *Given* some initial condition(s),
  - *When* event(s) occurs,
  - *Then* ensure some outcome(s)

Executable Criteria

- The specification is specified in a way it is executable
- Directly represented in code and used to exercise your application code

Tools for BDD

- Java - JBehave, JDave, beanSpec, Instinct
- *easyb*

easyb

- Started by Andy Glover
- Express Story and Spec using Groovy Based Domain Specific Language (DSL)
- Highly expressive
- Can be used for Java and Groovy applications

Writing a Story

- A Story file can contain any number of scenarios
- Each scenario has three parts: *given*, *when*, *then*
- when is optional

```
scenario 'text', {
 given 'text', {}
 when 'text', {}
 then 'text', {}
}
```

Use " instead of ' if you want to embed expressions in text

Writing a Story

- You can have more than one of *given*, *when*, *then*
- *When* is optional

```
scenario 'text', {
 given 'text', {}
 and
 given 'text', {}
 when 'text', {}
 then 'text', {}
}
```

Expressing Conditions—should

- You can verify values on **any** object using one of the following *should* constructs

```
shouldBe
shouldEqual
shouldBeEqual
shouldBeEqualTo
```

```
shouldNotBe
shouldNotEqual
shouldntBe
shouldntEqual
```

```
shouldBeA (like shouldBeAInteger)
shouldBeAn
shouldNotBeA
shouldNotBeAn
```

```
shouldHave
(for property of object or
member of collection)
```

```
account.balance.shouldEqual 10000
```

ensuring (asserting)

- You can ensure or assert values using closure syntax

```
ensure(!value)

ensure(value) { isFalse }
```

operates on value given to ensure

add multiple conditions using ***and***

```
isNull
isNotNull
isA<class type>
isEqualTo(value)
isEqualTo<value>
isNotEqualTo<value>
isTrue
isFalse
```

```
contains(member)

contains(property:val)

(for member of collection
  or properties of object)
```

Writing a Spec

- Specs/Behaviors start with `it`
- You can have as many of these you like in a Spec

```
it '', {
}
```

Story Example

file:money.story

```
scenario 'deposit money', {
  given 'account 12345'
  when 'deposit $50'
  then 'balance of account 12345 goes up by $50'
}
```

Unintegrated or Pending Story

Running Story

On my machine easyb is an alias to
"java -classpath ... org.disco.easyb.BehaviorRunner"

```
> easyb money.story
Running money story (money.story)
Scenarios run: 1, Failures: 0, Pending: 1, Time Elapsed: 0.549 sec

1 behavior run (including 1 pending behavior) with no failures
>
```

You can provide multiple story files to easyb

Fake Integration

```
scenario 'deposit money', {
  given 'account 12345', {}
  when 'deposit $50', {}
  then 'balance of account 12345 goes up by $50', {}
}
```

```
> easyb money.story
Running money story (money.story)
Scenarios run: 1, Failures: 0, Pending: 0, Time Elapsed: 0.578 sec

1 behavior run with no failures
>
```

Integration

```
scenario 'deposit money', {
  given 'account 12345', {
    account = 12345
    service = AccountService.create
    balance = service.getBalance(account)
  }
  when 'deposit $50', {
    service.deposit account, 50
  }
  then 'balance of account 12345 goes up by $50', {
    service.getBalance(account).shouldEqual balance + 50
  }
}
```

AccountService.java

```
public class AccountService
{
  int _balance = 100;
  public static AccountService getCreate()
  {
    return new AccountService();
  }

  public int getBalance(int account)
  {
    return _balance;
  }

  public void deposit(int account, int amount)
  {
    _balance += amount;
  }
}
```

Obviously a trivial example to get test pass, real AccountService will be talking to Account(s)

Running Story

```
> easyb money.story
Running money story (money.story)
Scenarios run: 1, Failures: 0, Pending: 0, Time Elapsed: 0.679 sec

1 behavior run with no failures
>
```

Let's Break It

```
public void deposit(int account, int amount)
{
  //_balance += amount;
}
```

```
> easyb money.story
Running money story (money.story)
FAILURE Scenarios run: 1, Failures: 1, Pending: 0, Time Elapsed: 0.605 sec
        "balance of account 12345 goes up by $50" -- org.codehaus.groovy.runtime
.InvokerInvocationException: org.codehaus.groovy.runtime.InvokerInvocationExcept
ion: VerificationException: expected 150 but was 100:

1 behavior run with 1 failure
>
```

Fix it and try again

A Narrative

```
description '''This is about depositing money into
checking accounts
'''

narrative 'description', {
as_a 'account holder'
i_want 'deposit money'
so_that 'whatever benefit...'
}

scenario 'deposit money', {
  given 'account 12345', {
```

Another Story

```
// appended to money.story
scenario 'deposit $10000', {
  given 'account 12345'
  when 'deposit $10000'
  then 'balance of account 12345 goes up by $10000'
  and
  then 'notify homeland security'
}
```

Running The Two Stories

```
> easyb money.story
Running money story (money.story)
Scenarios run: 2, Failures: 0, Pending: 1, Time Elapsed: 0.617 sec

2 total behaviors run (including 1 pending behavior) with no failures
>
```

Reports

```
usage: BehaviorRunner my/path/to/MyFile.story
 -txtspecification <file>   create a behavior report
 -txtstory <file>           create a story report
 -xml <file>                create an easyb report in xml format
```

```
> easyb money.story -txtstory stories.txt
Running money story (money.story)
Scenarios run: 2, Failures: 0, Pending: 1, Time Elapsed: 0.618 sec

2 total behaviors run (including 1 pending behavior) with no failures
```

Reports

file: stories.txt

```
2 scenarios(including 1 pending) executed successfully

  Story: money
   Description: This is about depositing money into
checking accounts

   Narrative: description
      As a account holder
      I want deposit money
      So that whatever benefit...

    scenario deposit money
      given account 12345
      when deposit $50
      then balance of account 12345 goes up by $50

    scenario deposit $10000
      given account 12345
      when deposit $10000
      then balance of account 12345 goes up by $10000 [PENDING]
      then notify homeland security [PENDING]
```

Other Options to Run

- Ant
- Maven
- IntelliJ IDEA

- Refer to http://www.easyb.org

Specifications

file: purchaseSoda.specification

```
vendingMachine = VendingMachine.instance

it "should dispense a can of Pepsi", {
   cans = vendingMachine.cans
   vendingMachine.purchaseSoda "Pepsi", 100
   vendingMachine.cans.shouldEqual cans - 1
}

it "should fail if you ask for Coke", {
  cans = vendingMachine.cans

  ensureThrows IllegalArgumentException, {
    vendingMachine.purchaseSoda "Coke", 100
  }
  vendingMachine.cans.shouldEqual cans
}
```

References

- http://behavior-driven.org
- http://jbehave.org/
- http://codeforfun.wordpress.com/gspec/
- http://www.easyb.org/

- Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans, Addison-Wesley.

You can download examples and slides from
http://www.agiledeveloper.com - download

Thank You!

Please fill in your session evaluations

You can download examples and slides from

http://www.agiledeveloper.com - download