



# GRAILS TESTING: DONE, DONE, DONE!

GEOFF WEBB ([gwebb@2paths.com](mailto:gwebb@2paths.com)),

2PATHS SOLUTIONS LTD.



2PATHS

# ... now it works (kinda)

---

- UNIT tests work in isolation (no spring autowiring)
- INTEGRATION tests glue things together (Spring, HSQLDB)
- `Bootstrap.config` gets slurped up correctly
- `GrailsUnitTestCase` out of the box convenience



# Specify what test you want to run

---

- *call specific test phases:*

```
grails test-app -unit
```

```
grails test-app -integration
```

- *run specific tests:*

```
grails test-app -unit MyDomain
```

runs tests in: `test/unit/MyDomainTests.groovy`

- *or rerun failing tests:*

```
grails test-app -rerun
```



# GrailsUnitTestCase : you need it!

---

- Mocks things for you conveniently

```
def strictControl = mockFor(MyService)
strictControl.demand.someMethod(0..2) {int i->
    return "things"
}
```

```
// I don't care about *** calling order!
def looseControl = mockFor(MyService, true)
```

- Metaclass magic handling:
  - `registerMetaClass(MyClass)` - for all your meta magic, cleaned up in `tearDown()` so as not to pollute VM



# mockFor...() methods

---

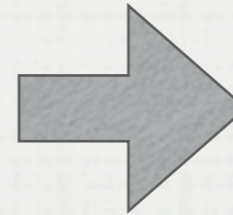
Nice wrappers for convenient mocking

- `mockFor()` - mock anything with expectations
- `mockDomain()` - add `save()`, `validate()`, `find*()` YAY!
- `mockForConstraintsTests()` - adds `validate()` for constraint tests
- `mockController()` - yup, those too!
- `mockTagLib()` - add `'out'`, `'render'` etc
- `mockLogging()` - add mock logger > stdout



# You test your domain objects ... right?

```
void testConstraints() {  
  
    def existingMember = new Member(  
        username: 'fred@example.com',  
        password: 'testing')  
  
    mockForConstraintsTests(Member, [existingMember])  
  
    // a member needs a username and password  
    def member = new Member()  
    assertFalse member.validate()  
  
    // username MUST be an email address  
    member = new Member(  
        username: 'fred',  
        password: 'anotherone')  
  
    assertFalse member.validate()  
  
    // username MUST be unique  
    member = new Member(  
        username: 'fred@example.com',  
        password: 'anotherone')  
  
    assertFalse member.validate()  
  
    // all things considered,  
    // we should have a valid member  
    member = new Member(  
        username: 'anne@example.com',  
        password: 'secret')  
    assertTrue member.validate()  
}
```



```
class Member {  
  
    String username  
    String password  
  
    static constraints = {  
        username(blank: false,  
            unique: true,  
            email: true)  
  
        password(blank: false)  
    }  
}
```



# ... and your services

```
def memberService

protected void setUp() {
  super.setUp()
  memberService = new MemberService()
}

protected void tearDown() {
  super.tearDown()
}

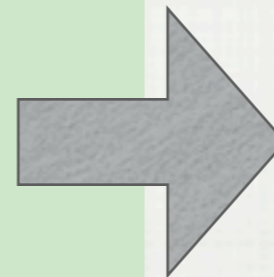
void testRegisterMember() {
  def existingMembers = []
  mockDomain(Member, existingMembers)

  // registration should call mailService.sendMail
  // lets mock out an impl of MailService
  def otherControl = mockFor(MailService)
  otherControl.demand.sendMail(1..1) { email, template ->
    return true
  }

  memberService.mailService = otherControl.createMock()

  // registering a member should result in it being
  // saved and an email being sent
  def member = new Member(username: 'user@example.com',
                          password: 'testing')
  def savedMember = memberService.registerMember(member)

  assertEquals 1, existingMembers.size()
  assertEquals member, savedMember
  otherControl.verify()
}
```



```
boolean transactional = true

def mailService

Member registerMember(Member member) {
  if(member.save()) {
    mailService.sendMail(member.username,
                          "welcome")
  }

  return member
}
```



# ... plus don't forget Controllers

```
protected void setUp() {
    // N.B. make sure we use our parents magic
    super.setUp()

    // we also need to mock any domain/commands
    mockDomain(Member)
    mockCommandObject(RegisterCommand)

    // mock our service layer
    // (it is a unit test after all)
    def mockControl = mockFor(MemberService)

    // just pass through what we get (i.e. always success)
    mockControl.demand.registerMember(1..1) { member ->
        member.save(); return member
    }

    this.controller.memberService = mockControl.createMock()
}
```

```
void testRegistrationSuccess() {

    def cmd = new RegisterCommand(
        username: 'test@test.com',
        password: 'test',
        passwordConfirm: 'test'
    )

    // just for good measure, make sure the cmd is correct
    assertTrue cmd.validate()

    // now test registration with a valid cmd
    controller.register(cmd)
    assertEquals('success', renderArgs.view)

    // also make sure we have the member
    def member = Member.findByUsername(cmd.username)
    assertNotNull member
}
```

```
def register = { RegisterCommand cmd ->

    if(!cmd || cmd.hasErrors()) {
        render(view: 'register',
            model: [command: cmd])
    }
    else {

        def member = memberService.registerMember(
            new Member(username: cmd.username,
                password: cmd.password))

        if(!member || member.hasErrors()) {
            render(view: 'register',
                model: [command: cmd,
                    member: member])
        }
        else {
            render(view: 'success')
        }
    }
}
```



# More interesting base classes!

---

- **UNIT** testing:
  - **ControllerUnitTestCase** - unit test controllers, yay!
  - **TaglibUnitTestCase** - nicities for taglibs (mocks rails methods)
  
- **INTEGRATION** testing:
  - **WebFlowTestCase** - those pesky webflows
  - **GroovyPagesTestCase** - page rendering/GSP/taglibs



# Tricks, tips and caveats

---

- Command objects  $\neq$  Domain objects when it comes to constraints
- \*UnitTestCase** are meant for unit testing - i.e. no auto Spring malarchy; You still need some integration tests
- work in progress, test the test framework and send feedback
- docs not always accurate : "use the source luke!"



# Done!

---

- <http://grails.org/doc/1.1/guide/9.%20Testing.html>
- [https://svn.codehaus.org/grails/branches/GRAILS\\_1\\_1/src/groovy/grails/test/](https://svn.codehaus.org/grails/branches/GRAILS_1_1/src/groovy/grails/test/)
- <http://stateyourbusiness.blogspot.com/2008/08/unit-testing-controllers-with-testing.html>

`echo $feedback > gwebb@2paths.com`

