

Improving web application automation testing
with Groovy, Spock and Geb
Petyo Dimitrov, Musala Soft

Agenda

- Previous project experience
- Identified problems
- Introduce the testing framework
 - Selenium / Web Driver
 - Groovy
 - Spock
 - Geb
- Demo



a Starfleet science officer chats up an Egyptian goddess in a jazz club on the Moon

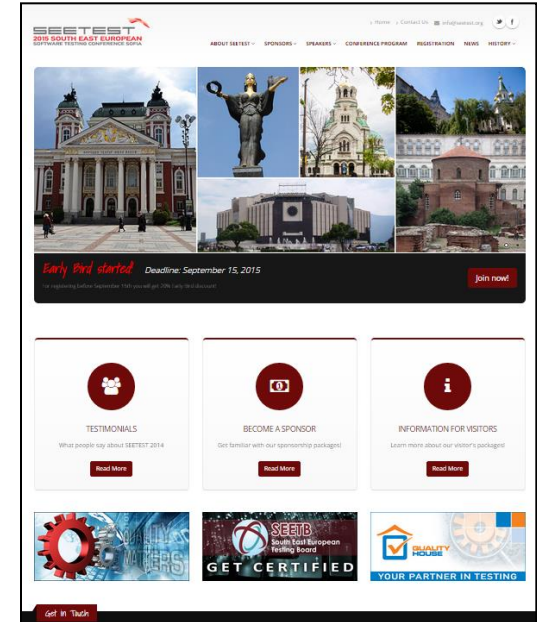
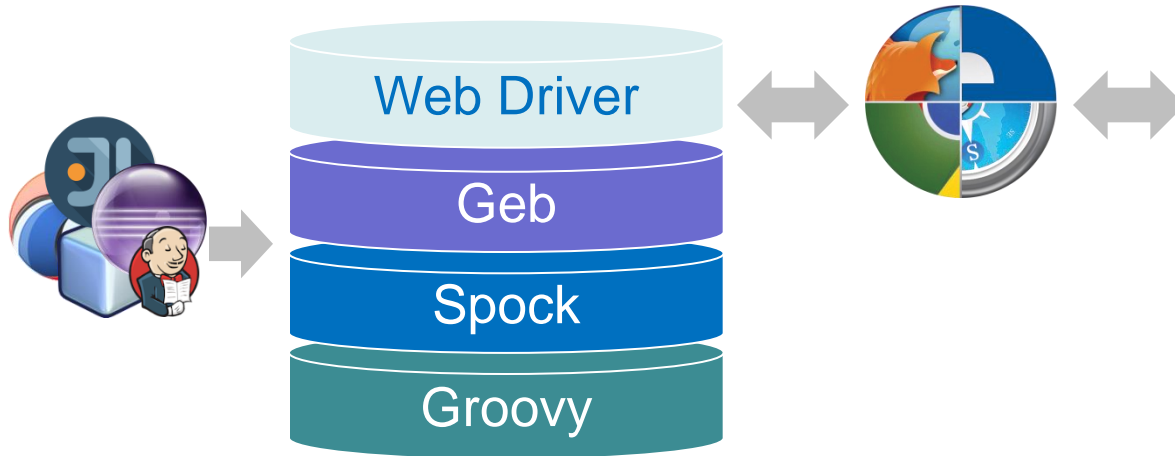
Previous project experience



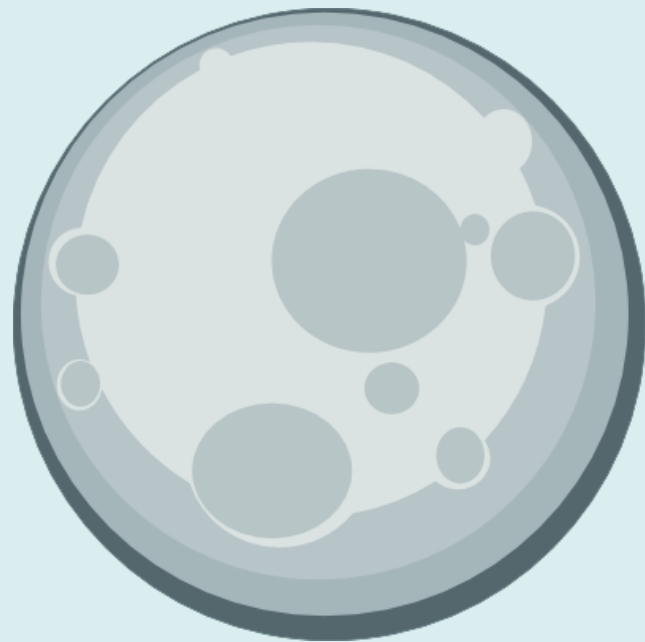
Problems

- different frameworks for DEV and TEST – no knowledge sharing
- duplication of code between DEV and TEST – no code reuse
- frameworks with high learning curve

Solution



Selenium Web Driver



Selenium Web Driver

- successor to the Selenium project
- offers cross browser automation
- API becoming W3C standard

Selenium Web Driver – example

```
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.*;
WebDriver driver = new FirefoxDriver();
driver.get("http://google.com");
WebElement search =
    driver.findElement(By.name("q"));
searchBox.sendKeys("webdriver");
driver.findElement(By.name("btnK")).click();
```

Selenium Web Driver – pros/cons

Good:

- 1.Proven solution for cross browser automation
- 2.Stable API and feature set
- 3.Active development

Bad:

- 1.Verbose code
- 2.Not a complete solution
- 3.Low level (Java invocations, XPath selectors)

Groovy



Groovy

- dynamic object-oriented language
- concise syntax – write less code faster
- compatible with Java ~ zero learning curve
- executes inside the JVM
- supports functional programming features
- powerful collections API
- suitable base for defining DSL-s

Groovy – concise

```
public class King {  
    private String name;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String threaten(String target) {  
        return name + ": Surrender your weapons king " + target + " or  
I will destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
King king = new King();  
king.setName("Xerxes");  
System.out.println(  
    king.threaten("Leonidas"));
```

Groovy – concise (semicolon)

```
public class King {  
    private String name  
    public String getName() {  
        return name  
    }  
    public void setName(String name) {  
        this.name = name  
    }  
    public String threaten(String target) {  
        return name + ": Surrender your weapons king " + target + " or  
I will destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
King king = new King()  
king.setName("Xerxes")  
System.out.println(  
    king.threaten("Leonidas"))
```

Groovy – concise (parentheses)

```
public class King {  
    private String name  
    public String getName() {  
        return name  
    }  
    public void setName(String name) {  
        this.name = name  
    }  
    public String threaten(String target) {  
        return name + ": Surrender your weapons king " + target + " or  
I will destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
King king = new King()  
king.setName "Xerxes"  
System.out.println  
    king.threaten("Leonidas")
```


Groovy – concise (public & return)

```
class King {  
    private String name  
    String getName() {  
        name  
    }  
    void setName(String name) {  
        this.name = name  
    }  
    String threaten(String target) {  
        name + ": Surrender your weapons king " + target + " or I will  
        destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
King king = new King()  
king.setName "Xerxes"  
System.out.println  
    king.threaten("Leonidas")
```

Groovy – concise (optional types)

```
class King {  
    private String name  
    String getName() {  
        name  
    }  
    void setName(name) {  
        this.name = name  
    }  
    String threaten(String target) {  
        name + ": Surrender your weapons king " + target + " or I will  
        destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
def king = new King()  
king.setName "Xerxes"  
System.out.println  
    king.threaten("Leonidas")
```

Groovy – concise (properties)

```
class King {  
    String name  
  
    String threaten(String target) {  
        name + ": Surrender your weapons king " + target + " or I will  
        destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
def king = new King(name: "Xerxes")  
println king.threaten("Leonidas")
```

Groovy – concise (gstrings)

```
class King {  
    String name  
  
    String threaten(String target) {  
        "${name}: Surrender your weapons king ${target} or I will  
        destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
def king = new King(name: "Xerxes")  
println king.threaten("Leonidas")
```

Groovy – concise summary

```
public class King {  
    private String name;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String threaten(String target) {  
        return name + ": Surrender your weapons king " + target + " or  
I will destroy your farms, slay your people, and raze your city."  
    }  
}
```

```
King king = new King();  
king.setName("Xerxes");  
System.out.println(  
    king.threaten("Leonidas"));
```

*Surrender your weapons king Leonidas or I will destroy
your farms, slay your people, and raze your city.*



*Come and get them,
Xerxes*

Groovy – base for DSLs

- DSL = Domain Specific Language
- Why is groovy suitable?
 - flexible syntax: . and () are optional*
 - access to the code's AST
 - meta-programming through `ExpandoMetaClass`

```
take (2.pills) .of (vitamins) .after (6.hours)
```


Spock



Spock

- unit testing framework
- for Groovy and Java applications
- highly expressive DSL
- has BDD and DDT features
- compatible with many IDEs and CIs
- inspired by JUnit, JMock, Mockito,...

Spock – example

```
import spock.lang.*
```

```
class UniformSpec extends Specification {
```

```
    def "uniform is blue by default" () {
```

```
        setup:
```

```
        def uniform = new Uniform()
```

```
        expect:
```

```
        uniform.color == "blue"
```

```
    }
```

```
}
```

```
public class Uniform {
```

```
    public String getColor() {
```

```
        return "blue"
```

```
    }
```

```
}
```

Spock – failure example

expect:

```
uniform.color == "green"
```

Condition not satisfied:

```
uniform.color == "green"
```

```
|           |           |
|           blue false
|           4 differences (20% similarity)
|           (blue) e (-)
|           (green) e (n)
```

```
com.musala.muffin.spock.Uniform@6cf145b5
```



Spock – feature method blocks

setup:

expect:

when:

then:

cleanup:

where:

- initializes the fixture
- must be at the start of a method
- must not be repeated
- implicit – all code up to the first label
- **given:** can be used instead

Spock – feature method blocks

setup:

expect:

when:

then:

cleanup:

where:

- combines stimulus and response
- may contain only conditions and variable definitions
- suitable for no-side effect functions

Spock – feature method blocks

setup:

expect:

when:

then:

cleanup:

where:

- always used together
- express stimulus and response
- response may contain:
 - conditions (including exceptions)
 - interactions
 - variable definitions

Spock – feature method blocks

setup:

expect:

when:

then:

cleanup:

where:

- cleans allocated resources
- mirror of **setup:**
- invoked event in case of exceptions

Spock – feature method blocks

setup:

expect:

when:

then:

cleanup:

where:

- parameterizes feature method with test data
- must be last
- uses `<< []` to supply list of values
- uses `|` and `||` to supply table

Spock – example (blocks)

@Unroll

```
def "'#role officers' wear '#color'" () {
```

```
    def crewMember = new CrewMember()
```

```
    when: "a crew member is assigned to a role"
```

```
    crewMember.assignTo(role)
```

```
    then: "the uniform is changed accordingly"
```

```
    crewMember.uniformColor == color
```

```
    where:
```

```
    role << ["science", "security"]
```

```
    color << ["blue", "red"]
```

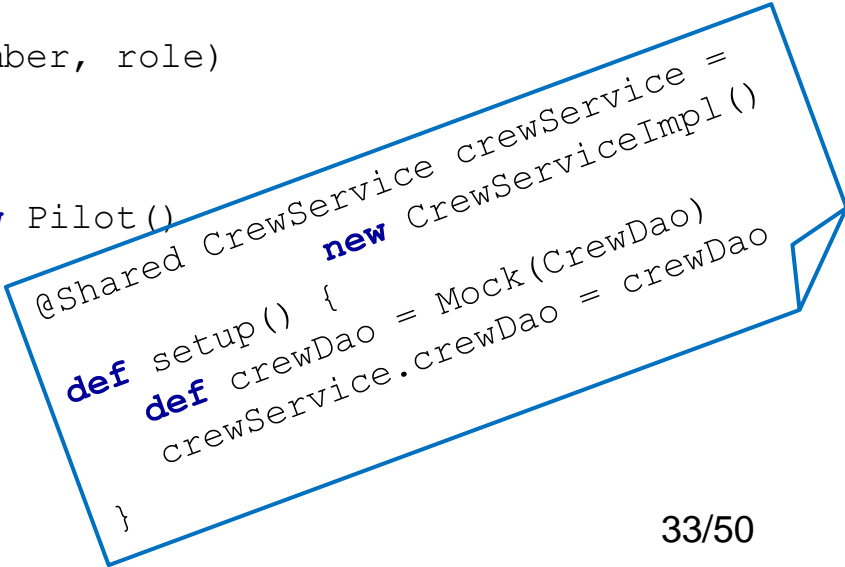
```
}
```

execution summary:

- '#role officers' wear '#color'
- 'science officers' wear 'blue'
- 'security officers' wear 'red'

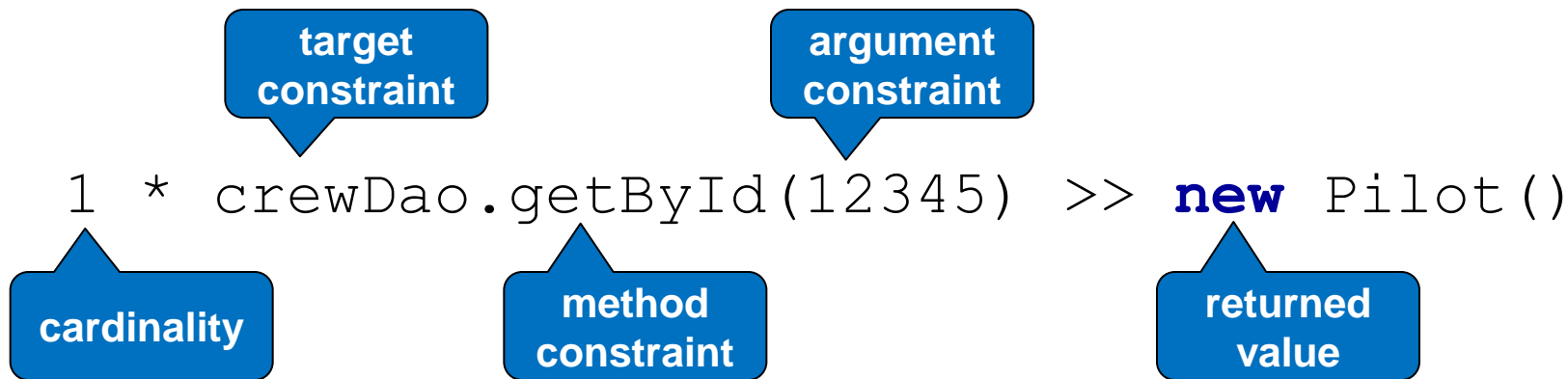
Spock – example (mocking)

```
def "assigning crew member to unknown role fails" () {  
    String crewMember = "123"  
    String role = "science"  
  
    when:  
        crewService.assignMemberToRole(crewMember, role)  
  
    then:  
        1 * crewDao.getById(crewMember) >> new Pilot()  
        0 * crewDao._  
        1 * roleDao.getById(role) >> null  
  
        EntityNotFoundException ex = thrown()  
        ex.message == "role does not exists"  
}
```



```
@Shared CrewService crewService =  
    new CrewServiceImpl()  
  
def setup() {  
    def crewDao = Mock(CrewDao)  
    crewService.crewDao = crewDao  
}
```

Spock – interactions



```
0 * crewDao._  
0 * _
```

Geb



Geb

- browser automation solution
- combines:
 - power of WebDriver
 - elegance of jQuery selectors
 - robustness of Page objects
 - expressiveness of Groovy
- can be used for UI testing and scraping

Geb – navigator API (1)

`$("css selector", index, <attribute matcher>)`

`$("h2")` → all headings

`$("h2.test")` → all headings with class test

`$("h2", 0)` → first heading

`$("h2", 0..3)` → first four headings

`$("h2", id: "subject")` → heading with id

`$("h2", id: "subject", text: "muffin")`

Geb – navigator API (2)

```
$ ("h2", text: ~/mu.+/)
```

```
$ ("h2", text: startsWith("mu"))
```

```
$ ("h2", text: endsWith("in"))
```

```
$ ("h2", text: contains("ff"))
```

```
$ ("h2").previous(), $ ("h2").next()
```

```
$ ("h2").siblings("div")
```

```
$ ("h2").parent(), $ ("h2").children()
```

Geb – script usage

```
Browser browser = new Browser()
browser.go "/system/login"

assert browser.page.title == "Login page"
assert browser.page.find("h2").text() ==
"Sign in"

...

browser.quit()
```

Geb – page objects, why?

- pages isolate implementation details/change
- pages prevent duplication

- what does this code do?

```
$ ("input[name=un1]").value("qwerty")
```

```
$ ("input[name=pd3]").value("qaz")
```

```
$ ("input[type=submit]").click()
```

Geb – page example

```
class LoginPage extends Page {  
  static url = "/system/login"  
  static at = { title == "Login page"}  
  static content = {  
    username { $("input[name=un1]") }  
    password { $("input[name=pd3]") }  
    submit { $("input[type=submit]") }  
    captcha(wait: true) { $("img.cap") }  
    message(required: false) { $("div") }  
  }  
}
```

Geb – modules

```
class SearchResult extends Module {  
    static content = {  
        title { $("div.main").text() }  
        description { $("div.info").text() }  
    }  
}  
  
class SearchResultsPage extends Page {  
    static content = {  
        searchBox {module SearchBox, $("input.search") }  
        results {moduleList SearchResult, $("table tr") }  
    }  
}
```

Geb – test example

@Stepwise

```
class LogicSpec extends GebReportingSpec {
```

```
  def "display login page"() {
```

```
    when: "I go to login page"
```

```
    to LoginPage
```

```
    then: "Captcha is present"
```

```
    at LoginPage
```

```
    captcha.exists()
```

```
  }
```

```
}
```

```
  def "attempt login with incorrect pass"() {
```

```
    when: "I enter wrong credentials"
```

```
    username = "user"
```

```
    password = "incorrect-pass"
```

```
    submit.click()
```

```
    then: "Login fails with error message"
```

```
    at LoginPage
```

```
    message == "incorrect credentails"
```

```
  }
```

Geb – waiting

- `waitFor` allows waiting for a condition
- configurable, with default values

```
waitFor { $("div.error") }.text() == "Error"  
waitFor(timeout: 10, retry: 0.5) {...}  
waitFor("slow") {...}
```


Geb – interactions

- simple DSL syntax
- based on WebDriver Actions

```
interact {  
    keyDown ALT  
    clickAndHold <content-id>  
    moveToElement $("div.container")  
    keyUp ALT  
    release()  
}
```

Geb – configuration

- GebConfig.groovy

```
driver = {  
    new FirefoxDriver() // "firefox"  
}  
  
waiting {  
    timeout = 2  
    slow { timeout = 100 }  
}  
  
reportsDir = "reports"
```

Geb – areas for improvement

- IDE support
- dynamic DSL limitations
- browser dependent CSS3 support
- keyword collisions for content (page, config, driver)



**KEEP
CALM**

IT

**WORKS ON
MY MACHINE**

Thank You!

Petyo.Dimitrov@musala.com

Associate Software Architect, Musala Soft

Q&A

