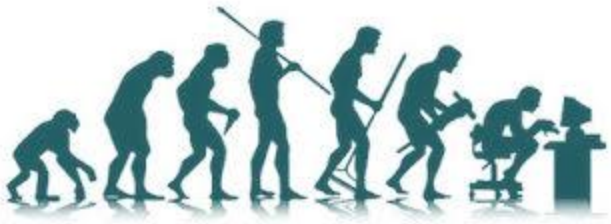
SpecRunner

[available at www.specrunner.org]

thiago.lvl.santos@gmail.com



Questions about testing?

- Process?
  - Formal Vs “informal”: “its all about fashion!”
  - Agile implies TDD (ATDD)?
- Premises & Problems?
  - Target audience? Performance? Maintenance? ∞
- Demands?
  - Outsourcing? Metrics? Legal restrictions? ∞
- Tools?
  - Depends on... everything. Where are we?
    - Lets consider TDD! Black or White.
  - This is not about stress or load testing... but data for these tests should be easy to produce.

Wish list

- Good support ☺;
- Multiple inputs types;
- Free specification style;
- Replaceable components;
- Compositional components;
- Better performance (time & memory);
- Better reports (useful and nice looking);
- Better error messages;
- Customized expression language;
- Predefined fixtures for things that matters;
- Event aware (listeners);
- Built in object reuse support;
- Thread-safe and support for concurrency issues;
- Real-time debugging;
- Reports for developers and for end users;
- Easy to extend;
- Easy to integrate;
- JUnit friend (good?);
- Spring friend (good?);
- Maven friend (good?);
- Lessons learned...

Learning from...

Concordion

- Site:
  - http://www.concordion.org
- Pros:
  - Documentation (web site);
  - Learning curve;
  - Free specification style;
  - Adoption;
  - Ports to other languages.
- Cons:
  - HTML input only;
  - Poor meta-variables set;
  - Fixtures for every project ("But I'm not the only one")

Fit + Fitnesse/SLIM

- Site:
  - http://fit.c2.com/
  - http://fitnesse.org/
    - (FitNesse.UserGuide.SliM)
- Pros:
  - Documentation (Guides);
  - Predefined actions (+SLIM);
  - Multi-input formats;
  - Wiki style (PO are writers?);
  - Adoption.
- Cons:
  - Poor API;
  - Performance (-SLIM);
  - Setup.

Learning from...

JBehave

- Site:
  - http://jbehave.org/
- Pros:
  - Documentation (web site);
  - Specification language;
  - Multiple reports;
  - Highly configurable.
- Cons:
  - Adoption;
  - Highly configurable.

Cucumber

- Site:
  - http://cukes.info/
- Pros:
  - Documentation (book);
  - Specification language.
- Cons:
  - Ruby (port to Java available)

Plain and match by regexp

```
Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON
```

Learning from...

Twist

- Site:
  - http://www.thoughtworks.com/products/twist-agile-testing/
- Pros:
  - Specification style;
  - Integration with Eclipse;
  - Support for refactoring.
- Cons:
  - Proprietary

Thucidides

- Pros:
  - http://thucydides-webtests.com/
- Pros:
  - Graphic reports;
  - Web testing.
- Cons:
  - Version 0.9;
  - Web testing only.

Learning from...

Robot framework

- Site:
  - http://robotframework.org/
- Pros:
  - Born in Nokia;
  - Variables;
  - Support for macros;
  - Predefined sets of fixtures.
- Cons:
  - Phyton/Jhyton;
  - Rigid specification style (table);
  - Setup.

∞ others

- JUnit:
  - Runners
- DBUnit:
  - DB restrictions
- HtmlUnit/Canoo Web Test:
  - Headless,GUI vs HTTP level
- JWebUnit:
  - HtmlUnit abstraction
- WebDriver (W3C):
  - Multi-target
- Selenium/Robot:
  - Fragile, record and play
- Jubula:
  - Swing? Web?

Specification audience

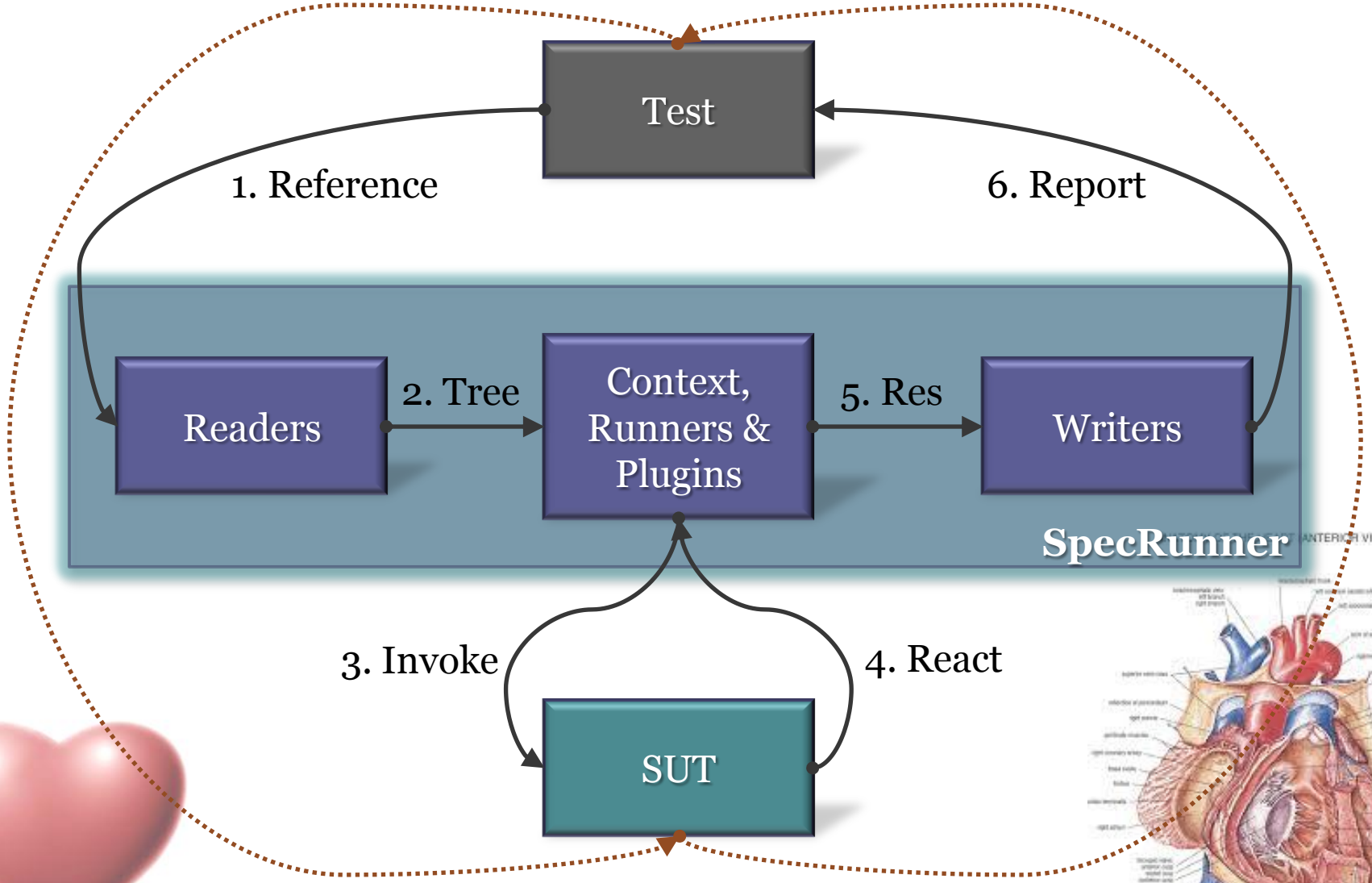
By Thomas Sundberg

That's all you need!

Design - Premises

- Old school (“one rule to rule them all” = hide)
  - “I know what is best for you”;
  - “Attributes must be private”;
  - “Friendly instead of protected”;
  - Private inner classes and so on.
- Tool/Framework ≠ End user product/API
  - Protected is very welcome;
  - Everything is replaceable;
  - Everything can be overridden;
  - Assume the consequences of your actions.

Design - Overview



Black box

White box

Calling SpecRunner

- Using SRRunner (+runners, introduces $THIS)
  - Test class become a fixture and there must be a HTML or Excel or TXT (...) file corresponding to it.

```
@RunWith(SRRunner.class)
public class TestSet {…}
```

- Point to a file (+runners):
  - Test class has methods @Test calling SpecRunner services.

```
public class TestSet {
    @Test
    public class run() {
      Iconfiguration cfg = …;
      SpecRunnerJUnit.defaultRun("input.txt","output.html", cfg);
    }}
```

- Don't use a specification file, create a program:

```
public class TestSet {
    @Test
    public class run() { …
      IPluginGroup group = …
      PluginBrowser browser = …;
      group.add(browser);…
      SpecRunnerJUnit.defaultRun(group, cfg);
    }}
```

Readers [HTML+ Excel + txt + ...]

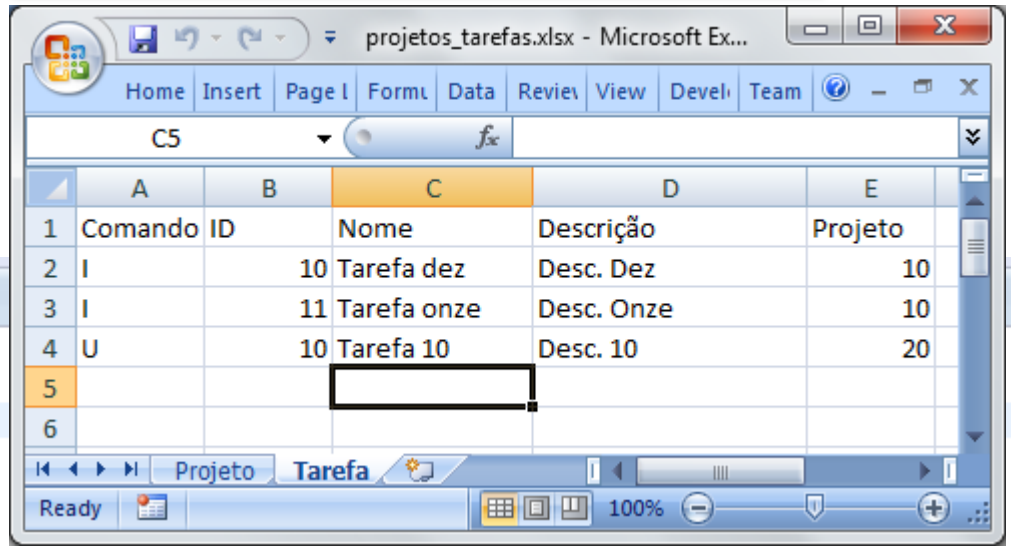
Specification is/as a tree!

HTML at specrunner-core

TestBasic.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="ISO-8859-1">
5  <title>Select examples</title>
6  </head>
7  <body>
8      <p>
9          Open <a class="include" href="../o
10     </p>
11
12     <ul>
13         <li>The word <span class="contains
14         <li>The word <span class="notConto
15         <li><span class="present" by="xpat
16         <li>None <span class="notPresent"
17         <li>There is only <span class="pre
18         <li>Check title that might be '<sp
19         <li>Date information might be '<sp
20     </ul>
21 </body>
22 </html>
```

Excel at specrunner-core-excel



Gherkin at specrunner-core-text (i18n)

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

Basic concepts – context & plugins

```
<html><body>...
  <p class="macro" name="googleSearch">
    When I open a <span class="browser">browser</span>,
    at <open>http://www.google.com</open> and search
    for <span class="type" by="name:q">#{Query}</span>,
    the result is #{Result}.
  </p>
  <p>
  Examples:
  <table class="map" name="examples“ scope=“body”>
    <tr><th>Query</th><th>Result</th></tr>
    <tr>
      <td>SpecRunner</td>
      <td>
      <span class="compareText" by="xpath://a[0]“>www.specrunner.org</span>
      </td>
    </tr>
  <table>
  </p><iterator name="examples“><call name="googleSearch“/></iterator>
</body></html>
```

CSS defines a macro named ‘googleSearch’.

Element open the given url.

Placeholder for value.

Placeholder accept nodes.

Defines a map named ‘examples’.

Placehoder names.

Lookup for xpath in result page

Call the macro for each example.

Basic concepts – core elements

- Plugin factories (CSS, element, attribute, text,..);
- Var (local, global, scoped, strict);
- Macros (definition and call);
- Maps (tables of named values);
- Loops (for, while, iterator);
- Conditions (ifs thens elses);
- Parametrized includes (multi-format);
- Expressions factories:
  - Values, classes instances, and models
- Configurations (local) Vs Features (global)
- Objects constructors (stubs support)

What is a plugin?

```
void initialize(IContext context)
    throws PluginException;
    Set plugin properties.

ENext doStart(IContext context,
   IResultSet result) …
    Perform on start action;
    Return skip or step into.

void doEnd(IContext context,
    IResultSet result) …
    Perform on end action.
```

Usage:

(CSS)

```
<span class="pause"
        enter="true"
        message="Pause!">
    <type by="…">…</type>
</span>
```

(Element)

```
<pause time="1000"/>
```

```
public class PluginPause extends AbstractPlugin {
  private boolean enter;
  public Long time;
  private String msg = "Pause requested.";
  // enter, SET/IS
  public void message(String message) {
      msg = message;
  }

  void initialize(IContext context) ... {
    super.initialize(context);
    IFM fm = SRServices.getFeatureManager();
    fm.set(FEATURE_ENTER,this);
    if(time == null){ fm.set(FEATURE_TIME,this); }
  }

  ENext doStart(Icontext context,…)…{
    if(time != null) { Thread.sleep(time); }
    else {
      if(enter){ pressEnter(msg); }
      else { showDialog(msg); }
    } return ENext.DEEP;
  }
}
```

Binding plugins

- Property files (loaded from classpath,multi-module)
  - **sr_plugins_css.properties**
    - pause=org.specrunner.plugins.core.flow.PluginPause
  - **sr_plugins_element.properties**
    - pause=org.specrunner.plugins.core.flow.PluginPause
- Plugin by example (on test setup)

```
IPluginFactory factory = SRServices.get(IPluginFactory.class);
PluginInclude increase = new PluginInclude();
increase.setDir("src/test/java/example/include");
increase.setHref("increase.html");
factory.bind(PluginKind.CSS, "incr", increase);
PluginInclude decrease = new PluginInclude();
decrease.setHref("excel/decrease.xlsx");
factory.bind(PluginKind.ELEMENT, "decr", decrease);
```

Usage:

```
<p class="incr">increment</p>

<decr/>
```

What about “fixtures”?

TestSet.java

```
@RunWith(SRRunner.class)
public class TestSet {
 public String greetingFor(String firstName) { return "Hello “+firstName+ "!“;}

 public void doSomethingOn(@Converter(args = { “MM/yyyy" }) Date date) {…}

 @Sentence(“with \\d+ and $string”) @Synonyms({“find $int and (\.+)”})
 public void search(int id, @Converter(args = { “MM/yyyy" }) DateTime date) {…}
}
```

TestSet.html (.htm,.xhtml)

```
<html>
  <head><link href="concordion.css" rel="stylesheet" type="text/css”></head>
<body>
    <p>
      The greeting for user <span class="set" name="firstName">Bob</span>
      will be: <span class="eq" value="$THIS.greetingFor(firstName)">Hello Bob!</span>.
      <sentence>Do something on: “12/2013”</sentence>.
      <sentence>Find users with 15 and 12/2013</sentence>.
    </p>
</body></html>
```

CSS/JS/images preserved.

Meta-variable

Automatic match and conversion.

Expressions (Janino at your service)

```
IExpressionFactory ef = SRServices.get(IExpressionFactory.class);
ef.bindValue("pattern", "HH:mm:ss").bindClass("dt", DateTime.class);
ef.bindModel("text6000", new IModel<String>(){
    public String getObject(IContext context) … {
        StringBuilder sb = …; for(int i=0;i<6000;i++){ sb.append(i%10); }
        return sb.toString();
    }});
```

Usage:

```
<span class="type" by="id:textArea">${text6000}excedente</span>
<type by="name:startDate" value='dt.toString(pattern)'>date</type>
<type by="id:xmlField"
      value='$INNER_XML'><date>${dt.toString(pattern)}</date></type>
```

- Meta-variables
  - $THIS
  - $BEAN
  - $NODE
  - $PLUGIN
  - $TEXT
  - $XML
  - $INNER_XML
  - $CONTENT
  - $CONTENT_UNSILENT
  - $PACKAGES
- Converters
  - sr_converters.properties
    - Primitives;
    - Dates;
    - Objects.
- Comparators
  - sr_comparators.properties
    - Primitives;
    - Dates;
    - Objects;
    - Nodes.

Writers [HTML+ console + ...]

Increase

SpecRunner details [success(13) in 38 ms]:
at 2013-11-10 13:40:10.024

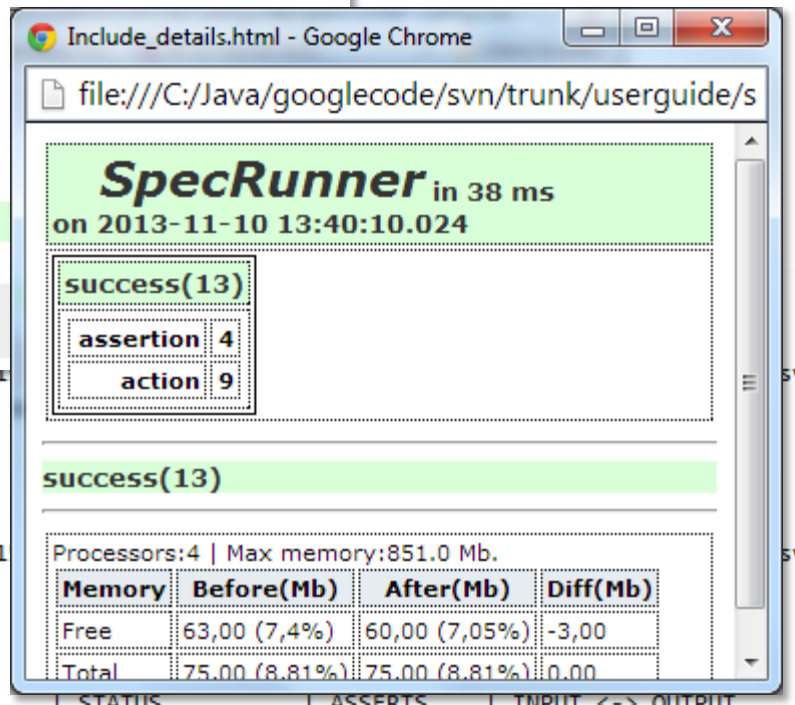
first include relative in a <A> tag

increase.html

increase counter

count is "1"

Decrease




```
                    1 |         614 |   100,00 | 2013-11-10 14:09:54.91
               ---------------------------------
                    TOTAL :         614 (AVG: 614,00)
    +-----------------------------------------------+
    +---------------- TXT (PERCENTAGE ORDER)---------------------+
                    # |   TIME (ms) |        % | ON
                    1 |         614 |   100,00 | 2013-11-10 14:09:54.91
               ---------------------------------
                    TOTAL :         614 (AVG: 614,00)
    +-----------------------------------------------+
    +---------------- TXT (STATUS ORDER)---------------------+
                    # |   TIME (ms) |        % | ON
                    1 |         614 |   100,00 | 2013-11-10 14:09:54.915
               ---------------------------------
                    TOTAL :         614 (AVG: 614,00)
    +-----------------------------------------------+
    +------ STATISTICS ------+
     NUMBER OF TESTS: 1
          TOTAL TIME: 614 ms
     FORMATED TIME: 00:00:00.614 (HH:mm:ss.SSS)
      AVERAGE TIME:  614,00 ms
            STATUS: [success=1]
             TYPES: [assertion=4, action=9]
    +-----------------------+
+-------------------------------------------------------------------------------+
```

```
| STATUS          | ASSERTS | INPUT <-> OUTPUT
| success 13/13   | 4/4     | C:\Java\googlecode\svn\
```

Do exist tests without assertions?

Coverage ≠ Correctness!

Profiles (sets of predefined plugins)

- Jetty + Tomcat:
  - Embeddable web servers.
- Hibernate3 + JPA:
  - Objects persistence and assertions.
- SQL (JDBC based):
  - High performance database setup and assertions.
- HtmlUnit + WebDriver:
  - Web browser technologies.
- Ant
  - Ant support

Jetty + Tomcat

Plugins

- Start
  - Dynamic ports;
  - Target port provider.
- Stop
  - “Shutdown” servers

Outstanding

- Jetty loaded from a jetty.xml file (more flexible);
- Support for objects reuse keep server alive until JVM shutdown if the configuration file is the same in different tests.

Hibernate3 + [JPA]

Plugins

- Configuration
  - Based on a configuration...
- SessionFactory
  - Prepare sessions...
- Objects extension
  - Create objects...
- Insert
  - To insert...
- Update
  - To update...
- Delete
  - To delete...
- Select
  - To verify.

Outstanding

- None code is required to create objects, only if you want (embeddable support provided);
- Object attributes are compared using comparators by type. For example, if an object attribute is of type DateTime (from JodaTime), the ComparatorJodaTime will be used, moreover, time comparators can set tolerance for these comparisons;
- Assertions using objects are independent of insertion/update/delete order.

SQL (reusable and cached objects)

Plugins

- Connection
  - Datasources
- Scripts
  - DDLs, DMLs
- Schema
  - Meta model
- Database
  - Comands executor.
- Prepare
  - Setup operations
- Verify (positive)
  - Assertions
- Compare base (negative)
  - Assertion.
  - Configurable filter.

Outstanding

- Connections can be thread independent;
- Scripts can be performed in failsafe mode;
- Interfaces for connection providers, schema loaders, database services, sequence handlers, comparison filters;
- Support for explicit IDs specifications, implicit identities, implicit sequences, all together;
- Database actions against multiple databases (compositional);
- Database comparison reports meaningful.

HtmlUnit+WebDriver

Plugins

- Browsers for web applications;
- Easy to map;
- Easy to extend;
- Easy to change;
- Actions:
  - By component type
- Assertions:
  - By component type
- With WebDriver (!htmlunit):
  - Real-time monitoring of web application behavior
  - Pause on failure listener is a must have feature.

Outstanding

- Search strategy unification: easily extensible;
- Rich output with your CSS/JS/images;
- Automatic screenshots of SUT;
- High customizable fields and table comparison;
- Smart node comparison: no more missing images or other elements and attributes (also inside tables);
- Compare Excel report: compare your table specification (in HTML or Excel) with a downloaded Excel file.

Ant

Plugins

- Perform Ant call
  - Requires a build file;
  - Basedir can be set relative to test or by absolute path.

Outstanding

- If something is missing, use ANT.

Conclusions

- Tools must serve, not slave!
  - Use the way you feel happy, but remember to avoid bad smells.
- Better abstractions implies in happier test designers and, perhaps, developers:
  - Choose how to specify: hardly it wont be able to instrument.
- Better development support implies in less unhappier developers:
  - Errors are stressing to all, specially to developers, help them to find problems is a good thing.
- Concurrent testing depend on concurrent system:
  - Avoid static variables, they should be carefully used.

Next steps

- Heavy users (very welcome);
- Load and stress test data can be fully generated with the help of looped specifications;
- Built in commons configurations: packages of configurations can be part of customization, for example: automatic inclusion of time variables such as ‘d’ or ‘dt’ for JodaTime LocalDate and DateTime respectively, disable CSS/JS/images to speed up;
- Debug of tool flaws not captured by user guide examples;
- Creation of other profiles (i.e. h3270 terminal interaction, file handlers – Ant is enough?).