Automating Tactically and Strategically

Alan Richardson

- www.eviltester.com
- www.compendiumdev.co.uk
- @eviltester

TESTING ASSEMBLY 09/2017, Helsinki

Do you automate tactically or strategically?

Automating Tactically
allows us to move fast, and
target immediate needs

Automating Strategically allows us to gain budget and support for longer periods of work

We may believe we are automating strategically but are actually automating tactically

Combine tactical and strategic approaches to better our testing efforts.

Secrets of Successful Practical Automating

- Get work done
- Automate Flows
- Multiple Abstractions
- Abstract to libraries, not frameworks

Testing field argues about

- Testing vs Checking
- You can't automate testing

"Test Automation" is contentious

An Example

"Is what follows a Test?" ...

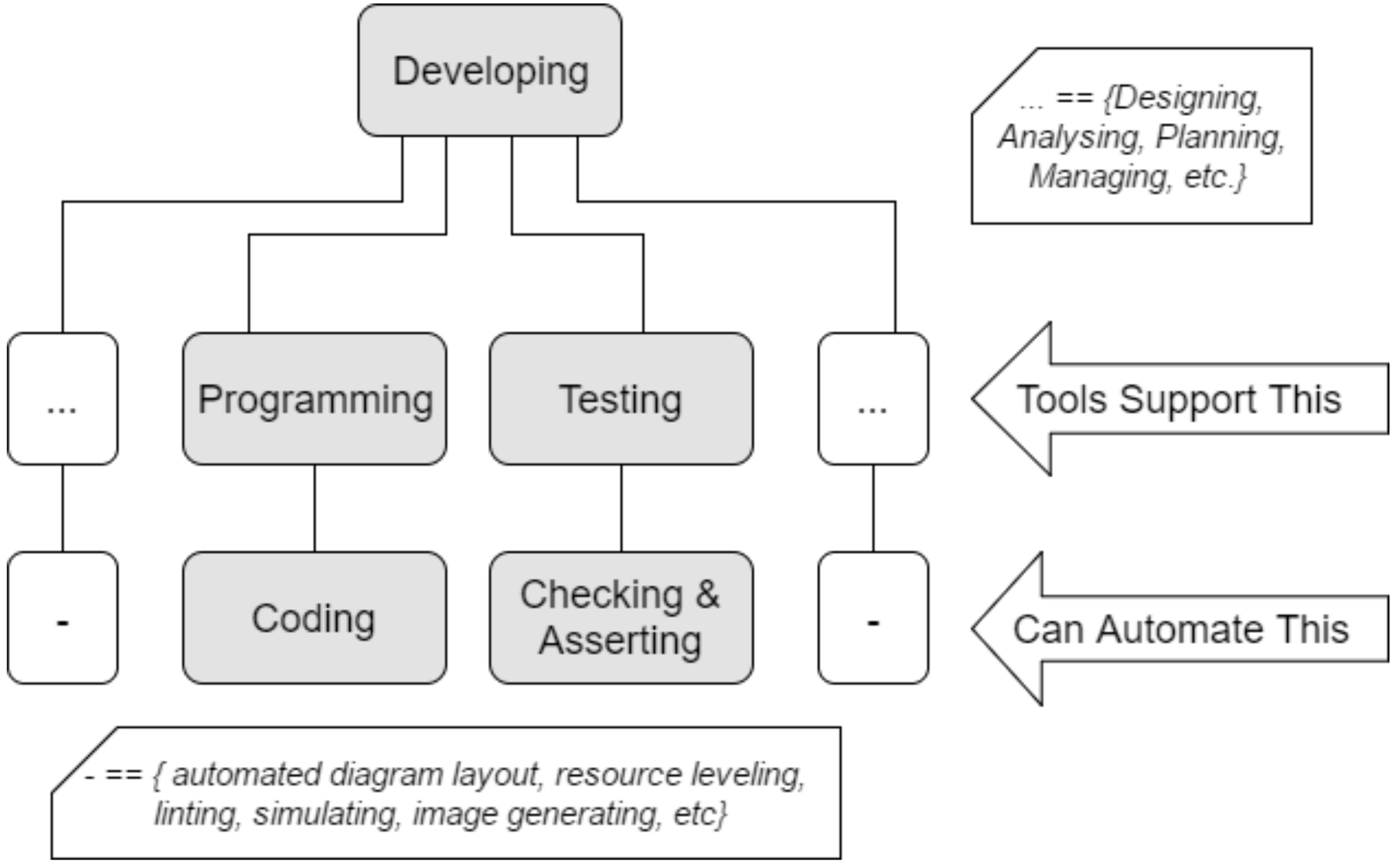
```
@Test
public void createPreviewMVP() throws IOException {

  String args[] = {
     new File(System.getProperty("user.dir"),
        "pandocifier.properties").getAbsolutePath()};

  new PandocifierCLI().main(args);

  String propertyFileName = args[0];
  File propertyFile = new File(propertyFileName);
  PandocifierConfig config;
  config = new PandocifierConfigFileReader().
              fromPropertyFile(propertyFile);

  Assert.assertTrue(new File(
                    config.getPreviewFileName()).exists());
}
```

An Example "Ce n'est pas un test", ...

- a Test, it says it is a `@Test`
- an Automated Test, it **checks** something (file exists)
- an Automated Test, it **Asserts** on the check
- a Tool - I use it to automatically build one of my books
- not an Application - it doesn't have a GUI or a standalone execution build

This is automated execution of a process

We Automate Parts of our Development Process: Coding, Checking, Asserting, ...


Developing
... == {Designing, Analysing, Planning, Managing, etc.}
...
Programming
Testing
...
Tools Support This
-
Coding
Checking & Asserting
-
Can Automate This
- == { automated diagram layout, resource leveling, linting, simulating, image generating, etc}

Tactical Reality - I don't really care, I just want to get work done

Suggests we also have 'strategic reality'

Tools vs Abstractions

- Gherkin
- Specflow, Cucumber

- - - - - - - - - - - - - - -

- Gherkin is a DSL templating language
- Specflow, Cucumber are template parsers
- We have to create abstractions to fill in the gap

E. W. Djkstra on Abstraction

"The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."

→ 1972 ACM Turing Lecture: 'The Humble Programmer'

Regression Testing

The system rarely actually suffers "Regression"

Continuous Condition Assertion

- TDD, ATDD, Automated execution
- All of these provide an ongoing safety net which might mean we don't need to revisit areas in the future (if we also tested them)
- They don't mean that we have 'tested' them now

Coverage - All coverage is Model Based

- Code coverage based on source code model
- What about requirements, system interfaces, states, flows?
- Code coverage is a useful metric for TDD coverage
  - coverage models based on risk

Tactical vs Strategic

Automating

Strategic does not mean 'strategy document'

Automating as part of a Testing Strategy

Automating Tactically vs Strategically

What is Tactical?	What is Strategic?
Solve a problem	Agreed & Understood Goals
Short Term	Long Term
Change as necessary	Slower to Change
Your Time	'project' time

Warning - sometimes tactics look like strategy

- Use of Jenkins is not Continuous Integration
- Cucumber/Specflow (Gherkin) != ATDD or BDD
- Automated Deployments != Continuous Delivery
- Containers != always good environment provision
- Page Objects != always good system abstractions

You know you are Automating strategically when...

- Reduced Maintenance & Change
- Everyone agrees why
- No fighting for 'time'
- It's not about roles
- It gets done

How do we automate?

- not about "test automation"
- it is about automating tasks
- automate the assertions used to check that the stories are still 'done'

Automate flows through the system

- vary data

- abstract the execution

Automating Strategically with Abstractions

see also "Domain Driven Design"

"modeling the system in code"

→ abstractions are about modelling - people seem to be scared of having too many models

No Abstractions - Web Testing Example

```
@Test
public void canCreateAToDoWithNoAbstraction(){
 WebDriver driver = new FirefoxDriver();
 driver.get("http://todomvc.com/architecture-examples/backbone/");

 int originalNumberOfTodos = driver.findElements(
             By.cssSelector("ul#todo-list li")).size();
 WebElement createTodo = driver.findElement(By.id("new-todo"));
 createTodo.click();
 createTodo.sendKeys("new task");
 createTodo.sendKeys(Keys.ENTER);
 assertThat(driver.findElement(By.id("filters")).isDisplayed(), is(true));

 int newToDos = driver.findElements(
                          By.cssSelector("ul#todo-list li")).size();
  assertThat(newToDos, greaterThan(originalNumberOfTodos));
}
```

But there were Abstractions (Lots of them)

- WebDriver - abstration of browser
- FirefoxDriver - abstraction Firefox (e.g. no version)
- WebElement - abstraction of a dom element
- createToDo - variable - named for clarity
- Locator Abstractions via methods e.g. `findElement`
- Manipulation Abstractions `.sendKeys`
- Locator Strategies `By.id`

Using Abstractions - Same flow

```
@Test
public void canCreateAToDoWithAbstraction(){
  TodoMVCUser user =
        new TodoMVCUser(driver, new TodoMVCSite());

  user.opensApplication().and().createNewToDo("new task");

  ApplicationPageFunctional page =
          new ApplicationPageFunctional(driver,
                                        new TodoMVCSite());
  assertThat(page.getCountOfTodoDoItems(), is(1));
  assertThat(page.isFooterVisible(), is(true));
}
```

What Abstractions were there?

- `TodoMVCUser`
  - User
  - Intent/Action Based
  - High Level
- `ApplicationPageFunctional`
  - Structural
  - Models the rendering of the application page

REST Test - No Abstractions

```
@Test
public void aUserCanAccessWithBasicAuthHeader(){

  given().
    contentType("text/xml").
    auth().preemptive().basic(
      TestEnvDefaults.getAdminUserName(),
      TestEnvDefaults.getAdminUserPassword()).
  expect().
     statusCode(200).
  when().
     get(TestEnvDefaults.getURL().toExternalForm() +
         TracksApiEndPoints.todos);
}
```

But there were Abstractions

- RESTAssured - `given`, `when`, `then`, `expect`, `auth`, etc.
- RESTAssured is an abstraction layer
- Environment abstractions i.e. `TestEnvDefaults`

Lots of Abstractions

But a lack of flexibility

Different abstractions to support flexibility

```
@Test
public void aUserCanAuthenticateAndUseAPI(){

  HttpMessageSender http = new HttpMessageSender(
                                    TestEnvDefaults.getURL());

  http.basicAuth( TestEnvDefaults.getAdminUserName(),
                  TestEnvDefaults.getAdminUserPassword());

  Response response = http.getResponseFrom(
                                   TracksApiEndPoints.todos);

  Assert.assertEquals(200, response.getStatusCode());
}
```

An API Abstraction

```
@Test
public void aUserCanAuthenticateAndUseAPI(){

  TodosApi api = new TodosApi(
                          TestTestEnvDefaults.getURL(),
                          TestEnvDefaults.getUser());

  Response response = api.getTodos();

  Assert.assertEquals(200, response.getStatusCode());
}
```

Automating API vs GUI

Automate at the interfaces where you can interact with the system.

Architecting to Support...

- Testing	- Support
- Deployment	- Automating
- Monitoring	- Devops
- Releasing	- etc.

Abstractions support different types of testing

- regular automated execution with assertions
- creation and maintenance by many roles
- exploratory testing
- stress/performance testing (bot example) requires thread safe abstractions

Good abstractions encourage creativity, they do not force compliance and restrict flexibility.

Supports Law of Requisite Variety

"only variety can destroy variety" - W. Ross Ashby

"only variety can absorb variety" - Stafford Beer

Frameworks can 'Destroy' variety

- Frameworks force us to comply with their structure
  - or things get 'hard'
  - or we don't reap the benefits from the Framework
- Some things are not possible
- Some things never occur to you to do

We have a lot of models

- user intent model - what I want to achieve
- user action model - how I do that
- interface messaging model - how the system implements that
- admin models, support models, GUI models

And models overlap

e.g. user can use the API or the GUI to do the same things

When we don't do this

- test code takes a long time to maintain
- test code takes too long to write
- test code is hard to understand
- test code is brittle - "every time we change the app the tests break/ we have to change test code"
- execution is flaky - intermittent test runs - "try running the tests again"

Example: Tactical Approach on Agile projects

- exploratory testing to get the story to done
- create a tech debt backlog for 'missing' automated tests
- `@Test` code is tactical and we can knock it up (not as good as production)

Example: Strategic Approach on Agile Projects

- automate assertion checking of acceptance criteria for "done"
- ongoing execution of the assertions for the life of the story

Example: Strategic Approach on Agile Projects implies...

- maintained for the life of the story in the application
- deleting the @Test code when no longer relevant
- amending code structure to make relevant as stories change and evolve
- @Test code is strategic asset (just like production code)

Why labour the point?

- ”test automation” often means ”testers do the automating”
- testers may not program as well as programmers
- testers may not know various programming patterns
- when programmers see `@Test` code written by testers they often don’t want to touch it or maintain it

Move from Abstractions such as 'Programming' and 'Testing' to 'Developing'

I say I'm a "Test Consultant"

I'm really a "Software Development Consultant"

Testing is Part of the Development process

Summary

- Automate Strategically
- ongoing assertion checking automated in @Test code
- Abstractions model the system in code
- Write abstractions at multiple levels
- Good abstractions can support exploratory testing
- Write good code all the time

Learn to "Be Evil"

- www.eviltester.com
- @eviltester
- www.youtube.com/user/EviltesterVideos

Learn About Alan Richardson

- www.compendiumdev.co.uk
- uk.linkedin.com/in/eviltester

Follow

- Linkedin - @eviltester
- Twitter - @eviltester
- Instagram - @eviltester
- Facebook - @eviltester
- Youtube - EvilTesterVideos
- Pinterest - @eviltester
- Github - @eviltester

BIO

Alan is a test consultant who enjoys testing at a technical level using techniques from psychotherapy and computer science. Alan is the author of the books "Dear Evil Tester", "Java For Testers" and "Automating and Testing a REST API". Alan's main website is compendiumdev.co.uk and he blogs at blog.eviltester.com (see also TesterHQ.com)