Test-Driven UI Development

OSCON 2012

TRADESHIFT®

Who, what, and why?

Joakim Recht

Senior Code Monkey at Tradeshift

“Deploying before lunch”

TRADESHIFT®

Common methodologies

- CEO-Driven Development
- Bug-Driven Development
- Test-Driven Development
- Behavior-Driven Development

A couple of words on BDD

- “Successor” of test-driven development
- Do more than just test your interface methods
  - Only test one aspect
  - Name the test properly
  - Explain what are the preconditions and the postconditions
  - Make the test readable by non-techs

The state of UI testing

Deeply unscientific

TRADESHIFT®

- Non-existing
- Non-maintained
- Non-maintainable

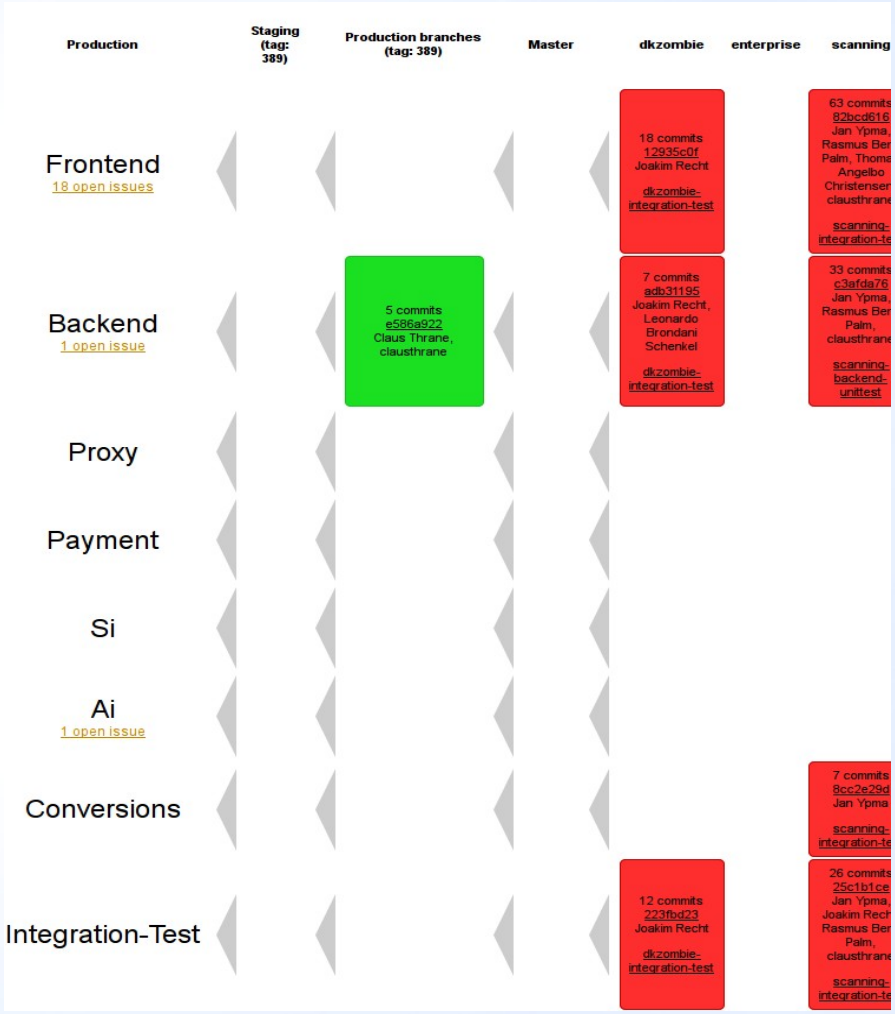


Wouldn't it be nice if...

- You actually had a UI test suite?
- The test suite was maintainable?
- You ran the tests automatically on all commits?
- You didn't deploy regressions?
- Everybody ran the tests?

The Deployment Pipeline

- All tests run automatically
- No merge to master if anything is failing
- No deploy if anything is failing
- Tests are optional, but only if there's a reason



TRADESHIFT®

I also want this... but how?

TRADESHIFT®

3 generations of UI testing

- 1st: Basic tests running, non-automated, messy
- 2nd: Larger coverage, automated, live view, got messy
- 3rd: Much more structure, even larger coverage, maintainable abstractions, parallelization

The main technologies

- Groovy (but your app doesn't have to be in Groovy)
- Selenium – remote-control browsers
- Geb – Write concise Selenium code
- Spock – Write understandable tests

Groovy

- Dynamic language for the JVM
- Mature and extremely powerful
- Easy to learn and write

  (also easy to create write-only code)

Selenium

- Java-based framework for remote-controlling a browser
- Can be used either for recording or for scripting
- Works with all major browsers
- Contains the Webdriver API

Geb

- Groovy-based framework on top of Selenium
- Makes it much easier to use Selenium
- Introduces more structure to tests
  - Page classes and modules
  - Uses Spock for specifications

Spock

- Groovy-based framework for writing tests
- Tests are specifications
- Tests become more than just a list of assertions
- Builds on Junit – executable by any CI

Let's see some code

TRADESHIFT®

That looked pretty simple?

Unfortunately, there's more

TRADESHIFT®

The real challenges

Maintainablility

Stability

Speed

Maintainability

- Write testable code
- Create code conventions for Javascript/HTML/CSS integration
- Treat test code as if it was your production code
  - Don't copy/paste
  - Create abstractions to reuse code
  - Do BDD

Stability

"Oh, the tests failed?
Try running them again"

- Be very careful about Ajax
- Don't blindly waitFor
- Introduce lifecycle hooks
- Establish clear rules for accessing HTML from tests
- Do not assume anything – all tests should be able to run in any system state

Most important:

Do not ignore transient failures!

Speed

- Running through a browser is inherently slow, so
  - Only execute what's necessary for the test
  - Use IDs rather than class names in page definitions
  - Abstract common operations into helpers, and allow these to access business logic directly
    - Through API or database
    - Creating new users, domain objects, etc.
  - Parallelize or run in the cloud

Stuff that's still not being tested

- Text copy & translations
- Layout and styling
- Interaction design
- Email interactions
- Crazy interactions – click 3 times on a button, paste in a Word document, increase font size by 500%, etc.

Thank you
Questions?

Geb: http://gebish.org

Spock: http://code.google.com/p/spock/

jre@tradeshift.com

@joakimrecht

http://gplus.to/recht

http://tradeshift.com/blog/

TRADESHIFT®