What We Test in Wikimedia

Amir E. Aharoni
@aharoni

Pune Language Summit
November 2012

What we have
What we lack
What can we learn

PHP

PHPunit – what we have

- Integrated with Gerrit using Jenkins – can't merge code if tests don't pass
- Comprehensive tests for the wiki syntax parser
- Coverage of other areas is partial
- Tests for some extensions
- CLDR plural rules – tested
- Gender and grammar – tested

PHPunit – what we lack

- i18n testing is partial
- Only unit tests
- Tests can only run properly on a MediaWiki installation in English
  - For example, they include some hard-coded English messages in the expected results
- This causes issues to be missed, because people try to make tests pass

JavaScript

QUnit – what we have 1

- Tests for some of the JavaScript in core MediaWiki
- Most extensions don't have JavaScript tests, even though many have a lot of JavaScript
- Our i18n extensions were among the first to have QUnit tests \o/
  - WebFonts, Narayam (keyboard layouts)

QUnit – what we have 2

- QUnit tests for our portable jQuery libraries:
  - ULS
  - i18n (parser and loader for MediaWiki-like L10n)
  - WebFonts (MediaWiki-independent)
  - IME (new generation of Narayam)
- Part of the the build process using Phantom.js – doesn't require a browser window
- Integrated with GitHub pull requests (Travis)

QUnit – what we lack

- Integration with Jenkins.
- As with PHPunit: tests may fail if the wiki language is not English
- Using some hacks we can switch the language during the test, but it's not really robust

Random stuff

Debugging

- We have a debug mode, which doesn't minify the JavaScript for easier testing and debugging
- Challenge: Unfortunately, for various reasons it also means that RTL auto-flipping (CSSJanus) is not done
  - Lesson: integrating minification and i18n-related processing may prove troublesome

Pseudo-localization

- We have a fake "RTL English" mode for debugging
- It's not actually used much – better to use real people who know an RTL language :)

Frontend

Not yet

- Some beginning attempts to use Selenium and Watir, but nothing seriously integrated yet

Humans

a.k.a PEBKAC

2009 – Vector

- A big upgrade to the default skin of Wikipedia
- UX testing was outsourced
- User testing with a few dozens of people in San Francisco
- Not regular editors of Wikipedia
- Results:
  - A lot of i18n bugs missed
  - A lot of veteran users complained about incorrect features

2012 – ULS

- Designed and performed by our interaction designers
- Tested – speakers of English, German, Russian, Hebrew, Breton, French, Hindi and other languages
- Over Skype and Google Hangout

2012 – ULS: prototypes

- Prototypes done in Photoshop, Inkscape, Illustrator, Pencil (a Firefox extension) and basic HTML/JavaScript/CSS
- Enough to show and test the workflow
- Incomplete implementation of the whole ULS logic
  - Just the main relevant scenarios
  - Scenarios that prove to be working are then implemented completely by the developers

2012 – ULS: process

- Users were asked to perform tasks:
  - Find your language (using any method)
  - Find a particular language (using any method)
    - Measure: which method is the most popular. (Answer: most people go for the search box.)
  - Find a language using the map
  - Find a language using the list
  - Find a language using the search box

2012 – ULS: results

- Confusion of new users helped identify and fix problems in the design.
- Experienced users suggested features to make their work more efficient
  - Example: Search by language code, rather than just language name (most people don't know ISO 639 codes, but power users do)

Dream:

Integrated i18n testing methodology

Known i18n problems

- Software translators translate without context – need better docs and screenshots
- Terminology may be inconsistent – need glossaries
- Functional keyboard shortcuts may collide with keyboard layouts in some languages

Known i18n problems, cont.

- Fonts may be too small
- Translations may be too long
- Translations created in run time may be wrong (gender, plural, grammar, concatenation surprises, RTL issues)

Currently

- Encourage developers to write documentation
  - Undocumented translatable messages are not supposed to pass code review, but in practice they often do
- Wait for translators to complain about bugs and unclear messages
  - This works quite well in the translatewiki.net community, but could always be more robust

The dream

- To have a methodology for making relevant documentation and testing procedures as automatic as possible:
  - Check coverage of all possible message permutations
  - Create all relevant screenshots for documentation
  - Warn about potential readability problems

Questions

?

Discussion

!

Thank you

:)