Software Testing, Quality Assurance and Maintenance Winter 2010

Lecture 27 — March 12, 2010

Patrick Lam *version 2*

QA for Web Applications

In this lecture, we'll talk about some of the opportunities and challenges in testing Web applications. You'll find references in-line.

Uninteresting cases. Let's clarify what I mean by web applications. Some things on the Web aren't interesting for us:

- static web pages;
- traditionally deployed server-side software, e.g. Joomla, which has suites of unit tests. (Such software needs testing too, but uses a traditional development model.)

Examples of Web Applications. Gmail, Google Docs, Reddit, Flickr, Salesforce, eBay.

(Question: How does e.g. Flickr differ from a private Gallery? There are similarities, but I claim that software development is a bit different because the Flickr developers can push new versions to users when they choose.)

Examples of Bad Web Applications. Strangely enough, many UW systems are bad web applications: Jobmine, Quest (historically; it's better now), GAP.

In my opinion: these systems have captive audiences and the users are not the purchasers of the system. ("Eat your own dogfood" is a useful, although not sufficient, principle for creating usable systems.)

Some properties that make these systems bad: poor UIs (require too many clicks to get something done), compatibility problems, poor performance.

Technologies. These systems usually have a multi-layer design.

- Client-side: HTML, Javascript/AJAX, Flash, Silverlight; Gears/HTML5 for heavier-duty client-side storage.
- Web server: Apache, IIS, etc.
- Middleware: EJB, PHP, Ruby on Rails, etc.
- Database: Oracle, MySQL, Postgres, etc.

Properties of Web Apps. Deploying software on the Web differs from traditional shrinkwrapped software in a few ways:

- served on-demand by web servers under your control;
- data is mostly stored server-side;
- heterogeneous user and client environment;
- users have more control over control-flow (e.g. Back button)

Implications for QA.

- release cycles are under your control;
- storing data is often your problem;
- scaling and security are concerns.

Note that modern desktop software also has some points in common with web applications these days.

Release cycles

The most relevant issue for this course is release cycles[1]. (We don't talk much about when to release in this course; traditionally, QA ought to be able to stop the release if there is a showstopper or too many less-critical bugs).

For a web application, you can release whenever you like. When should you release?

- don't release dataloss bugs or cause yourself denial-of-service problems;
- test to mitigate risk, but don't necessarily require perfection;
- experiment on users;
- "Release Early, Release Often".

In particular, you might want to release early, with few features and few bugs, and then continuously add features [2].

Coverage

So what about this whole coverage thing? The coverage criteria we've seen in class also apply to the code in web applications; we'd also need to take care to model the user-controlled control flow properly for system-level tests.

[1] `http://cacm.acm.org/blogs/blog-cacm/40796-frequent-releases-change-software-engineering/fulltext`

[2] `http://www.paulgraham.com/startuplessons.html`