Lecture 14: User testing, code tracing

CS 5150, Spring 2022
Logistics

• Schedule presentation with client/course staff
• Plan for user testing (report #4)
Lecture goals

• Improve future project progress and reporting
• Design interfaces within the constraints of web browsers and mobile devices
• Evaluate UI designs with user testing
Report #2 feedback
General comments

• Be proactive in eliciting requirements, evaluating designs
• Keep requirements verifiable
  • Under what conditions should the client accept that they have been met?
• Clarify functional requirements with scenarios, screenshots/mock-ups

• Use standard UML symbols appropriately
  • Actors
  • Nodes
  • Interfaces
• Identify components by looking for protocols
• Identify deployment environments
Examples

• CUPD
  • User story preambles
  • Stakeholder interviews
  • Design mock-up screenshots
  • Project schedule

• CMSX
  • User stories
  • Deployment and component diagrams

• Inline edits
  • Use cases
  • Deployment and component diagrams
Web and mobile interfaces

... continued from Lecture 13
Responsive design
Flexible grids

• Divide screen into columns
• Declare how many columns each element occupies at each breakpoint
  • Use more columns for narrower screens

• Example: Bootstrap

Aside: semantic markup
• Many attempts to make content, style separate concerns
  • HTML+CSS, LaTeX, DocBook XML, Content Management Systems
  • Allows content to be delivered in multiple media (web, print, ebooks)
• Tension with designing around content
  • Separating tightly-coupled info is more work, hard to maintain
  • Style rules tend to leak into content
Progressive enhancement

• Beware the fancy
  • Modern browsers are "evergreen" - they keep themselves up-to-date and support many of the latest web standards
  • But compatibility is still a concern
    • Support for standards is uneven (e.g. Edge vs. CMSX)
    • Mobile devices often stop receiving updates
    • User preferences, browser extensions, firewalls make browsers heterogeneous

• Progressive enhancement
  • Leverage fancier features to improve UX, but ensure that core functions are still available without them
  • Use fallbacks, polyfills to maximize compatibility
Poll: Progressive enhancement

PollEv.com/cs5150
Evaluation and user testing
Analyze/design/build/evaluate loop
Evaluation

- **Design** and **evaluation** should be done by different people
- **Schedule** must include time to conduct tests *and* make changes
- **Evaluation** should be ongoing
  - Iterative refinements during development
  - Quality assurance before deployment
  - Improvements after launch
- **Methods of evaluation**
  - Empirical (user testing)
  - Quantitative (measurements on operational systems)
  - Analytical (sans users; not in CS 5150)
Standards for usability: ISO 9241:11

• Effectiveness
  • The accuracy and completeness with which users achieve certain goals
  • Measures: quality of solution, error rates

• Efficiency
  • The relationship between the effectiveness and the resources expended in achieving them
  • Measures: task completion time, learning time, number of clicks

• Satisfaction
  • The users' comfort with and positive attitudes towards the use of the system
  • Measures: attitude rating scales
Poll: Measuring usability

PollEv.com/cs5150
User testing is time-consuming, expensive, and critical
Preparation

• Determine **goals** of usability testing
  • "*Can a user find the required information in no more than two minutes?*"

• Write the **user tasks**
  • "*Given a new customer application form, add a new customer to the customer database*"

• Recruit **participants**
  • Use the descriptions of users from the requirements phase to determine categories of potential users and user tasks
Participants

- Don't need many (per feature)
  - Diminishing returns after 5-6 users
  - Look for diversity (age, experience, ability)
- Combine structured tests with free-form interviews
- Have at least two evaluators per test
  - Should not include designers
- Advice: it's not a race!
  - Example: user testing for arXiv
Conducting sessions

• Environment
  • Informal
  • Simulated work environment
  • Usability lab
• Give the user their task
• Observe the user
  • Human observer(s)
  • Recording (with permission)
• Query satisfaction
Analyzing results

• **Test the system, not the users**
  • Respect the data and the user's responses
  • Do not make excuses for designs that failed
  • If possible, use statistical summaries

• Pay close attention to instances where users:
  • Were frustrated
  • Took a long time
  • Could not complete tasks

• Also note aspects of the design that *did* work
  • Ensures they are maintained / do not regress in final product
Example: Past CS 5150 methodology

How we're user testing:

• One-on-one, 30-45 min user tests with staff levels
• Specific tasks to complete
• No prior demonstration or training
• Pre-planned questions designed to stimulate feedback
• Emphasis on testing system, not the stakeholder!
• Standardized tasks / questions among all testers
Example

**Types of questions we asked:**

- Which labels, keywords were confusing?
- What was the hardest task?
- What did you like, that should not be changed?
- If you were us, what would you change?
- How does this system compare to your paper based system?
- How useful do you find the new report layout? (admin)
- Do you have any other comments or questions about the system? (open ended)
What we've found: Issue #1, Search Form Confusion!
What we've found:
Issue #2, Inconspicuous Edit/Confirmations!
What we've found:
Issue #3, Confirmation Terms
What we've found:
Issue #4, Entry Semantics
What we've found: Issue #5, Search Results Disambiguation & Semantics

<table>
<thead>
<tr>
<th>Select</th>
<th>Edit</th>
<th>Date/Time</th>
<th>Category</th>
<th>Type of Question</th>
<th>Medium</th>
<th>Staff</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>edit</td>
<td>2006-04-03 16:12</td>
<td>Reference</td>
<td>16 to 30 minutes</td>
<td>In Person</td>
<td>Librarian</td>
</tr>
<tr>
<td></td>
<td>edit</td>
<td>2006-04-03 16:04</td>
<td>Directional</td>
<td>1 to 5 minutes</td>
<td>In Person</td>
<td>Librarian</td>
</tr>
<tr>
<td></td>
<td>edit</td>
<td>2006-04-05 01:03</td>
<td>Reference</td>
<td>16 to 30 minutes</td>
<td>E-mail</td>
<td>Librarian</td>
</tr>
</tbody>
</table>
Measurement-based evaluations

• User testing can be done with (non-functional) prototypes
  • Requires more interaction with evaluator (risk of bias)
• Measurements require an operational system

• Log events in users' interactions with system
  • Clicks (when, where)
  • Navigation (from page to page)
  • Keystrokes
  • Use of help system
  • Errors encountered
  • Eye tracking
• May be used for statistical analysis or for detailed study of an individual user
Eye tracking
Analyzing measurements

- Which interface options were used?
- When was the help system consulted?
- What errors occurred? From where and how often?
- Which links were followed? (clickthrough data)

- Human feedback (less structured)
  - Complaints and praise in feedback forms
  - Bug reports
  - Calls to customer service
Refining designs

- Do not allow test evaluators to become designers
  - Designers are poor evaluators of their own work,
  - But designers know requirements, constraints, context of design
    - Know which problems might be addressed with small changes
    - Know which problems require major changes that should be escalated
    - Know which user requests are mutually incompatible
      - Balance between configurability and simplicity (designer's job)

- Designers and evaluators must work as a team
  - But not try to do each other's work
User testing in CS 5150

• All projects must conduct user testing of user interfaces you design
  • Internal projects: recruit classmates from other teams
  • Decide how much training users should have
    • They should probably be familiar with existing system
    • You can provide training (but don't "teach to the test"), or a user manual
• Design tasks & metrics
  • "Which files has your reviewer read so far?"
  • "Which, if any, of your commit messages has your reviewer left a comment on?"
  • "Add a reviewer comment to this file that was not modified"
• Design survey
Code tracing
Techniques

• Monitor application logs
• Developer tools network view
  • Look for mutating methods (POST, PUT, DELETE, vs. GET); ignore static resources
  • Look at initiator stack trace
    • Ignore framework methods (jQuery, etc.)
    • Look for promising files, then read them

• Search source code
  • Filter results (ignore static, tests, docs)