CS 307: Software Engineering

Lecture 11: Users, Usability, and Inspection

Prof. Jeff Turkstra
Announcements

- Design document due Monday, February 6
Lecture 11

- Users and Usability
- Defects and Inspection
User centered design

- Software development requires an understanding of...
  - your users
  - their tasks
- Users should be involved in the decision-making process
- User interfaces should strive for usability
How?

- Recruit users to test
- Collect feedback
  - Prototype
  - On-line help
  - Draft manuals
- Understand users’
  - Goals
  - Patterns of use
  - Demographics
  - Experience
Why?

- Reduced training and support costs
- Reduced time to learn
- Greater efficiency of use
- Reduced development costs
  - Know which features to focus on
- Easier to anticipate future changes
- Better prioritization
- User appeal on completion
UI Design basics

- Should be done in conjunction with other software engineering activities
- Use case analysis and user stories are valuable for defining the tasks that a UI should facilitate
- Iterative prototyping
  - Natural with scrum
Usefulness

- Do the actual uses of a product achieve the goals of the designers?
- Utility – ability of a product to perform a task or tasks
  - More tasks, more utility
- Usability – how easy is it to accomplish a prescribed task using the product?
Usability

- **Learnability** – how quickly can a new user become proficient with the system
- **Efficiency** – how quickly can an experienced user complete a task
- **Error handling**
  - Help prevent users from making errors
  - Detect errors when they occur
  - Correct them when possible
Likability

- Do not confuse likability with usability with utility
- More nebulous than other traits
  - Styling and flash
    - “Eye candy”
  - Conferred status
    - “I Am Rich”
  - How the user feels
  - Music and sounds
Usability

- **Discovery** – finding a feature in response to a need
  - How long does it take?

- **Learning** – time to complete a task with the feature

- **Efficiency** – to complete a task given a feature that has been mastered
Principles

1. Always test with users
   - Usability guidelines only get you so far

2. Identify tasks and design your UI around them
   - Use case analysis, user stories

3. Make tasks as simple as possible
   - Minimize reading and manipulation required by user
4. Give hints about what should be done next
   - Make sure users can see what commands are and are not available
   - Make important commands stand out

5. Provide clear and concise feedback
   - Keep users aware of their progress
   - ...and their location as they navigate
6. Effective error handling

- Explain the situation
- Help the user resolve it
  - When you can

7. Ensure users can get out, go back, and undo

- Allow operations to be undone
- Provide easy navigation mechanisms
8. UI should be uncluttered as much as possible

- Subjective, and dependent on task
- Organize information efficiently

9. Design for multiple user groups

- Different locales, support disabilities
- Consider the system from a beginner, intermediate, and expert standpoint
  - Consider providing an advanced or expert “mode” interface
10. Provide help

- Organize it well
- Integrate it with the application
- Keep it accurate

11. Be consistent

- Similar layouts
- Common graphic designs
- Typical look-and-feel standards
- Mimic other applications when appropriate
Evaluating UIs

- **Heuristic evaluation**
  - Subset of use cases or user stories
  - Try to pick the most important ones
- For each window, page, or dialog that appears while following the use case, study it in detail to look for possible usability defects
- When defects are discovered, write down
  - Short description
  - Proposed fixes
User observation

- Select users representative of most important actors
- Select most important use cases
- Write instructions for each scenario
- Arrange evaluation sessions
  - Explain their purpose
  - Video record if possible
  - Talk to the users as they perform tasks
  - Debrief after they are done
- Note difficulties and formulate recommendations for change
Failure

- A **failure** is an unacceptable behavior exhibited by a system
  - Frequency of failures $\rightarrow$ **reliability**
  - Achieving very low failure rate and high reliability is an important design objective
Defect

- A defect is a flaw in any aspect of a system that contributes, or may contribute, to the occurrence of a failure or failures.
  - Can be in requirements, design, and/or code.
  - May take several defects to cause a failure.
Common causes of defects

- **Poor communication**
  - Missing or misinterpreting information

- **Oversight**
  - Forgot to do something, skipped steps due to lack of time or other resources

- **Transcription error**
  - Typos followed by failing to read the artifact carefully

- **Lack of education or training**
  - Person was in over their head
Error

- Mistake or inappropriate decision by a software engineer leading to the introduction of a defect
Inspecting

- One or more people systematically
  - Examine source code or documentation
  - Look for defects
- Inspection meetings
- Desk inspecting

- Type of code review
  - Others include pair programming and walkthroughs
Inspecting and testing

- Inspections can start as soon as any document is generated.
- Test planning cannot start until the requirements and/or specifications have been written.
- Testing cannot start until the system components have been generated.
- Inspections examine static documents.
- Testing examines the actual product dynamically
- Inspection precedes testing
- Testing follows inspection
- Inspections find defects which can be missed by testing or are very hard (or too expensive) to find by testing
- Testing finds defects missed by inspections and verifies any performance requirements
- Both are needed to insure that we build quality products
Observation

- Fagan reported in 1986 that the defect insertion rate during the inspection correction process was one defect inserted for each six attempted defect removers.
Weinberg speculated in 1993 that a process with a fault feedback ratio (FFR) of 0.3 or greater is out of control.

$$\text{New Faults Inserted}$$

$$\text{FFR} = \frac{\text{Faults Found}}{\text{Faults Found}}$$
■ The cost of non-quality software typically accounts for 30% of the development costs and 50-60% of the life cycle costs
Studies indicate that each major error found by an Inspection will save at least four hours of downstream correction effort

- Some estimates range up to 30 hours

The estimated cost of running Inspections is about 10-15% of the development cost
Inspection steps

1. **Request:** initiating the Inspection process. Someone must volunteer an artifact.
2. **Entry:** screen out artifacts which will yield a low probability of successfully exiting the Inspection.
3. **Planning:** determine the objectives, schedule, resources, and tactics to be used on the artifact to be Inspected.
4. Individual checking: quality time spent by each checker searching and recording possible major and minor defects in the artifact under Inspection

- Each checker has a role – to look for special defects
- Picks their own time and place
- Works alone
- **Logging meeting:** collect and record issues noted by each checker. Search for additional problems as a group. Also a time to log ways to improve the Inspection process.

- **Edit:** issues found during Inspection are examined by the owners of the artifact and classified. Defects are hopefully corrected.
Follow up: Inspection leader checks to be sure all items addressed by the Inspection have been dealt with by the owners of the artifact

Exit: Inspection leader checks to be sure all steps have been followed, and that the Inspection data and metrics have been recorded
**Release**: artifact is released to the next software development stage along with an estimate of the number of possible major defects remaining.
Typical checking roles

- Usually 3 to 4 people, most hold multiple roles
  - **User** – customer’s point of view
  - **Tester** – viewpoint of testability, test plans, test requirements, etc
  - **System** – big picture, hardware requirements, manuals, documentation, schedule, other system needs
  - **Financial** – cost related implications, estimates, uncertainty, etc
- Quality – quality issues
- Service – installation, field service, customer service, maintenance
- Legal – defects that could create a legal hazard
- Safety – product causes us and the customer no harm
- Rules – rules and procedures are followed and appropriate
  - Code standard, etc
- **Backward** – works from back to front
  - Helps overcome fatigue
  - Latter parts of the artifact less thoroughly checked by others
Sample Inspection roles

- Error message checker
- Data type checker
- Function or procedure call checker
- Algorithm checker
- Logic checker
- Comment checker
- System call checker
- Code format and standards checker
- Macro and side effect checker
- Parameter testing and sanity test checker
- Initialization and use pairs checker
- Meets specification checker
- Meaningful name checker
What is logged?

- **Issues**
  - Critical
  - Major
  - Minor

- **Improvement suggestions**

- **Questions of intent to the author(s)**
What is not logged?

- Who found the defect
- Whether the issue is really a defect
- How to fix the defect
- Discussion of how the defect entered the artifact
### Inspection vs Testing

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Inspection</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module interface errors</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Excessive code complexity</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Unrequired functionality present</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Usability problems</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Performance problems</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Badly structured code</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Failure to meet requirements</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Boundary value errors</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Questions?