CS189A Section

Tuesday, November 8th 2022

Tobias Hollerer
Department of Computer Science
UC Santa Barbara
Overall Plan @Week 7

❖ Four 2-week sprints (dates adjusted)
  – Oct 10-23 (PRD v1 – tools, technologies, design, terminology)
  – Oct 24-Nov 6 (use cases/user studies, prototyping, PRD v1, PRD v2)
  – Nov 7-20 (design, prototyping, testing, PRD v2)
  – Nov 21-Dec 4 (prototype demo/pres prep, prototyping and testing)

❖ Fall presentations and demos: Dec 5, 4-7pm (Broida 1640)

❖ Specify what the product will do
  – Vision statement
  – Product Requirements Document (PRD) (due Oct 30 and Nov 28/29)
  – Design tools, brainstorming, coding (tests and implementation)

❖ Build and test an initial prototype
  – MVP
  – Typically teams iterate on these activities until they converge to a working prototype!
This Week’s Plan

❖ Team activities
  – Scrum: Sprint 3, Testing Suite, PRDv2
  – Complete “Mentor Connections” google survey

❖ Section: TA meetings

Upcoming deadlines:
❖ Nov 20: Testing Suite
  – Unit and Integration testing
❖ Nov 22: Sprint 3 ends (Sprint 4 starts)
  – Product Requirements Document v2
❖ Nov 28: PRDv2 due
Sprints 3 and 4

- Sprint 3 started yesterday
- Break down stories into tasks & components associated with design
  - Prioritize stories
  - Assign timings to stories/use cases AND store/use-case tasks
  - Specify acceptance/test that can be used to verify a story is finished
- Sprint: Prototype tasks (primary implementation before demo)
  - Prioritize tasks
  - Assign timings to tasks
  - Specify what test(s) are to be used as evidence of task completion/acceptance (use case post condition OR user story acc test)
  - Each member/developer chooses task, implements, and tests task
  - Another member does code review/test and accepts the pull request
    - Test is the one specified above (Acceptance)
  - When store/case is complete, some member performs story test/acceptance
Sample Story -
As a website user,

I need a “home” button

So that I can return to the homepage with a single click

Here are some acceptance criteria:

❖ Home button is visible on all pages
❖ Clicking the home button returns the user to the homepage
❖ Home button is recognizable
PRDv1: Your **Living** Requirements Document: A Shared Google Doc *(done)*

- Authors, Team, Project Title
- Intro – including problem, innovation, science, core technical advance (2-3 pages)
  - Define project specifics, team goals/objectives, background, and assumptions
- System architecture overview
  - High level diagram (1 page)
  - User interaction and design (1+ page)
- Requirements (functional and non-functional)
  - User stories or use cases (links) ✅ 10 for PRDv1 prioritized
  - Prototyping code, tests, metrics (5+ user stories): github commits/issues
- System models: contexts, sequences, behavioral/UML, state
- Appendices
  - Technologies employed
PRDv2: Your Living Requirements Document: A Shared Google Doc

❖ Authors, Team, Project Title

❖ Intro: problem, innovation, science, core technical advance (3+ pages)
  – Define project specifics, team goals/objectives, background, and assumptions

❖ System architecture overview
  – High level diagram (1 page)
  – User interaction and design (1+ pages) – ie detailed design

❖ Requirements (functional and non-functional)
  – User stories or use cases (links) □ 20+ for PRDv2 prioritized w/acc. tests
  – Prototyping code, tests, metrics (10+ user stories): github commits/issues

❖ System models (1+ pages)
  – Contexts, interactions, structural, behavioral (UML)
  – Use cases, sequencing, event response, system state, classes/objects

❖ Appendices - Technologies employed
Your Project Design: PRDv2

❖ Architecture (hardware/software)
   – Evolve your overview picture from PRDv1 to provide significantly more detail and any updates or changes

❖ Detailed design
   – UML diagrams of primary data structures that comprise the system architecture connected via their associations (if any)
     o Ensure that each "class" is balanced in terms of cohesion & coupling
     o Annotate with pre/post conditions when appropriate
   – Sequence diagrams
     o synchronous and asynchronous for key interactions between classes
       ▪ At least 3 different interactions
     o User interactions with the system
       ▪ At least 3 different interactions
       ▪ Can be a human user or a machine user (API) interaction
         » Event response, updated application state
       ▪ If you have a user interface: Provide mockups for primary UIs
PRDv2 User Stories / Use Cases

❖ Revise spec to add detail to the functional specification to match your design

❖ Add user stories and break up the stories you have into finer grained stories
  – Provide UML, sequence diagrams, dataflow diagrams
  – *Goal: a CS senior should be able to take your doc and implement the project*

❖ For each fine-grained story, provide a description and acceptance test
  – Provide time estimates (1 person-hours) for each story implementation
    o Ensure you can finish the implementation in the time you have (this/next quarter)
  – **Prioritize tasks** to have a complete prototype by the end of this quarter
    o Focus on the externally facing interfaces, **mock out** what you cannot get to
  – **Write unit tests** to implement tasks for mandatory tasks
    o Document these tasks (autogen the documentation/usage)
  – Add trello/pivotal task links (titles must match) to PRDv2 for each story

❖ Prototype designed mandatory tasks; **add github commit ID/link to PRDv2**
  – Github must have unit tests, documentation (for anything without unit tests), and prototyping implementations for each story in Sprint

❖ If you have a user interface
  – Provide mockups that are tied to the functionality described in 1+ components
Software Testing

❖ What and why

❖ How

❖ Planning

What is Software Testing

❖ Testing is the process of demonstrating that errors are not present

❖ The purpose of testing is to show that a program performs its intended functions correctly

❖ Testing is the process of establishing confidence that a program does what it is supposed to do

❖ Testing is the process of executing a program with the intent of finding errors
Most Costly Software Errors

- NASA’s Mars Climate Orbiter (1998, $125m)
- Ariane 5 Flight 501 (1996, $500m)
- EDS Child Support System (2004, $1b)
- Soviet Gas Pipeline Explosion (1982)
- Bitcoin Hack, Mt. Gox (2011, 850,000 bitcoins, $500m)
- Heathrow Terminal 5 Opening (2008, 10 days, 42000 bags, 500 flights)
- The Mariner 1 Spacecraft (1962, $18m)
- The Morris Worm (1988, $100m)
- Patriot Missile Error (1991, 28 deaths)
- Pentium FDIV bug (1994, $475m)
- Knight’s $440 Million Error (2012)
10 Principles of Testing

1. A necessary part of a test case is a definition of the expected output or result.
2. A programmer should avoid attempting to test his or her own program.
3. A programming organization should not test its own programs.
4. Thoroughly inspect the results of each test.
5. Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.
6. Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do.
7. Avoid throwaway test cases unless the program is truly a throwaway program.
8. Do not plan a testing effort under the tacit assumption that no errors will be found.
9. The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.
10. Testing is an extremely creative and intellectually challenging task.
Errors Remaining vs Errors Found: Surprising Relationship

Probability of the existence of additional errors vs Number of errors found already.
Black-box Testing vs White-box Testing

❖ Black-box strategy: data-driven, input/output-driven
   – Unconcerned about the internal logic
   – Test data derived from specification (user stories/use cases)
   – A criterion: exhaustive input testing

❖ White-box strategy: logic-driven
   – Test data derived from internal structure of software
   – Exhaustive statement coverage, path testing

❖ Management of complexity
Different levels of Software Testing

1. **Unit Testing (required)**: Individual units/components of a software/system are tested. Validate that each unit of the software performs as designed.

2. **Integration Testing (required)**: Individual units are combined and tested as a group. Expose faults in the interaction between integrated units.

3. **System Testing**: A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system’s compliance with the specified requirements.

4. **Acceptance Testing**: A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.
Unit test vs. Integration test
Testing Suite: Introducing Software testing

❖ Unit Testing
– Find a suitable testing library/tools for your platform.
– Test different units (functions/classes/modules).
– No minimum coverage requirement. (But use as a acceptance criteria for user stories)

❖ Integration/Component Testing
– Find a suitable testing library/tools for your platform.
– Test integration of at least one component.

https://www.tutorialspoint.com/junit/index.htm
A Bug Can Reveal a Missing Test

- ... but can also reveal that the specification was faulty in the first place, or incomplete
  - Code “evolves” and some changing conditions can trigger buggy behavior
    - Perhaps specification needs to evolve
  - This isn’t your fault or the client’s fault but finger pointing is common

- Great testing dramatically reduces bug rates
  - And can make fixing bugs way easier
  - But can’t solve everything: Paradise isn’t attainable in the software industry