Gathering Requirements

John Jannotti

course/cs0320/www/lectures/

Feb 23, 2017
Forming groups

- If you are already in a 4 person group, sit in the middle.
- If you have a 2 person group sit to your right: ⟷
- If you are 1 or 3 people, sit to your left: ⟷
Course Announcements

- It’s time to think about term project ideas.
- We will have a Piazza thread, all must submit an idea.
- Autocorrect is due in 4.5 days.
- Next week, lecture and lab will start talking about DBs.
- Another blood drive! Free Starbucks. 3pm-6pm.
- Society of Hispanic Professional Engineers - General Meeting. Tuesday 5pm. B&H 141.
Project ideas

What sort of project would you like to build?

A) Software for a “business” need. (reservation/appt system, employee scheduler, data analysis, tax preparation).

B) Software for individual productivity. (learning tool, stock tracking, contact management)

C) A game (Settlers of Catan, poker, chess, Scrabble, your own take)

D) Social software (match buyers/sellers, group event planning, book/movie recommendations)

E) Something else
Software Lifecycle

- Requirements Gathering
  - What does the user want to accomplish? Prioritize.
- Specification
  - Describe the software so that someone else could implement it.
- Software Design
  - Choose algorithms, data structures, architectures to divide the work allow clean implementation.
- Coding
- Testing
- Maintenance
  - Often by another team. Don’t make them hate you.
  - But it’s often you, revisiting code you wrote a long time ago.
A purely linear “lifecycle” has serious drawbacks.
- The customers’ needs are unclear and may change.
- It’s unclear what is difficult/easy to do.
- The best design may evolve once coding begins.

“Agile” development
- Iterates continuously. Weekly “releases.”
- Keeps the user involved in the process.
- We’ll demand many checkpoints, you’ll have to involve users.
Requirements

- Hopefully, you have a project idea.
  - What is the next step?
- You can’t start programming, or even designing yet.
  - First, you need to know what to build.
  - You need to understand your problem domain deeply.
  - You need to know what the users want.
Importance of Requirements

- Poor requirements cause project failures
  - 13% fail due to incomplete requirements.
  - 12% fail due to lack of user involvement.
  - 10% fail due to unrealistic expectations.
  - 9% fail due to changing requirements.
  - 7% fail because the system is no longer needed.

(Source lost to sands of time.)
An on-going process

- Take your first approximation seriously.
  - It will influence your basic spec/design.
  - Hardest to change.
- It will change for bad reasons
  - Can’t anticipate all the questions.
  - Users are fickle (or maybe you didn’t ask properly).
- And good reasons: Building the system creates opportunities.
  - Build a prototype \textit{fast}, get it in front of users.
  - If you prototype fast, you can throw it away.
  - Learn what matters most. Get better ideas.
- Iterate!
Purpose of Requirements

- Provide an understanding of users’ needs.
- How the system will be used.
- What the system has to do for users.
- Here is where a customer decides if they “got what they paid for.”
- Acceptance criteria (grades, too).
Gathering Requirements

- Identify **Stakeholders**
  - Multiple kinds of users. (manager, host, cook, waiter)
  - Entities affected, other than users. (payroll, employees)
  - Entities with a say (regulation, TAs).

- Observe and Interview
  - Observation of existing workflow
  - Competitive analysis
  - Interviews >> Surveys >> Role playing >> Nothing
  - User studies (on prototypes)
Documenting

- **Describe Scenarios**
  - Single use of the system.
  - Single feature or complete run.

- **Illustrate with Story boards**
  - Sequence of sketches of systems in action.
  - May include the UI, may include real-world actions.
Goals

- Define the problem from the viewpoint of users.
- Determine outline of the “best” solution.
- Determine what is required & optional.
  - Set priorities on features
- Determine limitations on resources
  - Team, target architecture, interacting systems...
- Determine acceptance criteria
  - Requirements should be testable (not the automated kind).
  - Requirements should be precise.
- Do not worry *much* about implementation.
Interview stakeholders

- Requires considerable preparation
  - Determine what information is needed.
  - Find out about interviewees – consider target demographics.
  - Decide on questions and organization.
  - An outline of talking points is insufficient. Think *Repeatable*.

- Process
  - Move from general to specific questions
  - Then end with very general questions
  - Process
    - Record Audio, Video
    - *Prefer Notes!* Revisiting audio/video is tedious.
    - Summarize: Stories, Use cases, Scenarios, Misconceptions
    - Follow-up: Keep contact info. You will be iterating.
Questionnaires

- Where there is a large set of users
- These are hard to develop as well
  - Opinion polling is big business, and yet polls disagree.
  - Try not to “lead the witness”.
  - But you want specific answers.

- Aim for questions with simple, but informative answers.
  - Not: Do you get book recommendations from friends? (uninformative)
  - Better: How do you find interesting books? (complex)
  - Best: How did you come to read your last book?

- Try to avoid bias, or “easy answers”
  - Would you like feature X? (The answer is always “Yes.”)
  - Better: Would you prefer X or Y?
  - Best(?): How would you like this to work?

- Wrap up with open-ended questions.
Observation

- Critical if you are automating an existing process.
- Competitive analysis is a form of this.
- Be unobtrusive.
- Avoid questioning early.
- Interview or Questionnaire \textit{later}.
- A user study is a special (contrived by you) case.
User Studies

- Ask users to conduct specific tasks.
  - Proceed as an observer (unobtrusive, avoid early questions)
  - Ask only for their running commentary. Keep them talking.
- You need (at least) a prototype.
  - Prototype could be (very) primitive.
  - Paper mockups can work.
  - Fake data (static screens)
  - Missing functionality (does the user even notice?)
  - “Wireframes” (keep discussion focused)
- Role playing as a degenerate case — pretend to be a user.
- Repeat as system matures.
- Humbling, and yet so useful.
  - You will think your users are dumb.
  - Then you will realize that you are.
Continuous Requirements

- Requirements analysis is an ongoing process
  - Initially you get a set of use cases
    - These will not fully cover requirements.
    - These are not even internally complete.
    - They will probably be inconsistent.
    - View these as a starting point

- Keep the user involved
  - Agile requirements
  - A/B Testing
Term Project Requirements

Who are your Stakeholders?
- You must have a real consumer, or many standins.
- Depends on target audience, but not just your group!
- Document your process. We should be able to repeat it.

Describe the system as the user sees it
- Scenarios describing functionality
- Survey/interview to get user input

Acceptance criteria
- What do you think constitutes passing? A-level work?

One to four pages
This will also help us match you to a TA.
Anecdote time!

- SightPath: SneakyBoot, Automatic topology formation
- Foodler: Red and Pretty, Best Bets
- A Few Billion Lines of Code Later
  - Nice account of engineers selling to “engineers” (and still having trouble nailing the requirements).
  - http://tinyurl.com/yb4rpc9
Your turn.

Volunteers to describe their project, identify stakeholders, talk about how you might gather requirements?