Requirements

Idaho State University | Computer Science

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will be able to:

- Understand the role that requirements and requirments gathering plays in software engineering
- Develop a goals document
- Construct both use cases and user stories to describe user requirements
- Understand how domain analysis interacts with requirements

Inspiration

"The hardest part of the software task is arriving at a complete and consistent specification, and much of the essence of building a program is in fact the debugging of the specification." – Fred Brooks

"If you cannot grok the overall structure of a program while taking a shower, you are not ready to code it." – Richard Pattis

Engineering Design Phases

There is a standard design methodology whenever you want to build buildings, devices, etc.

1. Get the **requirements** the e.g. bridge is supposed to meet
2. Make a complete **design** (blueprints, materials, building method, etc)
3. Build (**implement**) it
4. **Test** it to make sure its going to work.

In software engineering this is the classic waterfall model, it was the initial approach from which other approaches evolved.

Software Engineering Phases

Modern software engineering is a far more flexible variation on the above due to zero materials cost

- Start with a first pass at requirements and design which will not be complete.
- Implement the most-key features and get basic functionality running
- As an initial implementation takes shape, use it to refine the requirements and design
- Implement more features, continue to refine requirements/design more, and provide continual releases of the running application.

Schools of Software Development

- There are many schools of thought on the best way to develop software as a team
- We are not going to explicitly mention any of the schools here but you might want to read up on them.
- We are following some hybrid of the Agile, Extreme Programming (XP), and Scrum schools.

Requirements Gathering

Step one in building software: understand what the requirements are. This phase is also called Product Discovery.

- Software projects involving n people start out with n very different impressions on how the app should be structured.
- Your primary goal is to expand on and unify those n visions into one cohesive vision
- This happens through extensive discussing, questioning, sketching
- All parties involved need to be at the table: management, developers, customers.
- (For your projects, you may want to start with multiple ideas and work a bit on each one and pick a winner)

The Goal Documents

Here are some useful “goal” documents – if you can make these documents sing you have a unified vision!

1. a Vision statement - a paragraph or two on the shared vision of customer and developers on what the app should do.
2. a feature list - yes, a bulleted list of features the application should have
3. Sketches/storyboards of user interfaces and how they are used
4. Some use cases - for involved processes, a step-by-step sequences of how a particular scenario of user-app interaction should play out
5. Possibly some prototypes to verify libraries/frameworks will do what you need

Other Goals

- Identify the potential different parties (actors) that will be interacting with the system, and focus on their needs when gathering requirements
- **Domain analysis**: get a better understanding the underlying domain of the application (e.g. music file formats and copyright rules for a music streaming app)
- Explore different frameworks and libraries that could be used, and settle on an initial plan.

Other Goals

There are also bigger picture topics to keep in mind

- What is the market for the app? Who will actually use it?
- What is the feasibility of coding it in the available timeframe? Can it logically be split amongst the team?
- Look at the competition: do similar apps exist?
- What is the team organizational structure? Strong leader, collection or leaders, or highly democratic?

We now go through the goal documents in more detail.

Feature Lists

- List features the app should have, e.g. “Support Facebook login”, etc
- **core** features are the most basic ones, and extended features are long-range goals
- We will look at some feature list examples in lecture

Use-cases

(See Chapters 2-3 of the HFOOA&D book)

Use-cases (also called stories) are sequences of events that represent behaviors the functioning system should have.

Use-cases are helpful in fleshing out an application:

- The process of writing use-cases will expose details of expected functionality;
- You can incrementally elaborate and expand on (or, shrink and remove) use-cases until they are stable and sensible enough to implement.
- You can use the use-cases to help get started with coding.
- Here in requirements they are “customer-focused”, on how the system will be used, and evolve in design to “code-focused”, a sketch of an algorithm.

Examples of Use Cases

Chapters 2-4 and 10 of HFOOA&D are filled with examples of use-cases and their refinement.

- We will review use-cases in a previous OOSE project.
- One of the most complete examples is in the book on p. 124

Finding the Right Use Cases

- Use the feature lists, GUI sketches, etc as fodder for rooting out the use-cases.
- (HFOOA&D p. 78 and pp 304-305 give examples of extracting use-cases from feature lists)
- Focus on the meaty cases with a non-trivial number of steps, the short ones don't need writing out.
- Similarly, if a UI sketch or storyboard clearly shows the sequence of events, there is no need to duplicate that in a use-case.

Features vs use-cases

Features are single-sentence descriptions of features; use-cases are step-by-step descriptions of behavior. Here is a definition-by-example:

- A feature of a library book checkout application could be simply “The ability to charge late fees for overdue books”.
- A use-case of the same application could be “return book and calculate late fee”, a step-by-step scenario of how
  1. a patron returns a physical book,
  2. the book would be scanned,
  3. it would be marked as back on the stacks in the database, and
  4. a late fee if any would be computed and added to the patron’s account.

The Format for Writing Use Cases

To standardize projects we will ask that you follow the enumerated list format on p. 560 of the textbook; the above-mentioned p. 124 example also uses that format.

- Always include a **Title** which is a capsule summary
- The body is the main path of the use-case, a 1. 2. 3. .. numbered list of actions described in English
- Optional **Alternate Paths** of the use-case as per p. 124: other ways the use case could proceed (from p.124, take either 2. then 3., or take 2.1 then 3.1).
- Optionally, include an **Optional Path** – some steps that could be skipped (steps 6.1-6.5 in the p. 124 example)
- You don't need to use alternate or optional paths, include them if they give a more clear specification than making separate use-cases.

UI Sketches

- You don’t have to make them pretty, but it does help make the app feel real if you put some time into them
- Make sure to carefully go through features and use-cases to make sure you covered the features/uses with sketches.
- UI prototyping and storyboarding tools let you sequence a path through the different UIs, this can be a help

Initial Architecture Proposal

See the HFOOA&D book pp. 310-... and 501-...

- This is an initial proposal for the software architectures, frameworks, etc and how they will be deployed
- You don't have to stick with what you start with but stick something in for each piece you need.
- For example if your app includes face recognition in pictures and you wanted to use some third-party library to do that, list which library.

Domain Analysis

(See HFOOA&D book pp. 307-308)

You need to thoroughly understand the domain of your app before you can even begin to design it.

- Domain Analysis is any activity which increases the understanding of the underlying domain the software is working in.
- Example domains that need analyzing in detail:
  - In a restaurant (even without a computer) the "domain" is the actual restaurant with entities including `Orders`, `Menu Items`, `Tables`, `Payments`, `Receipts`, `Waitron`, etc.
  - In any implementation of a board/card/etc game there are the detailed rules, e.g. en passant moves in chess.
  - If you are making a social network app, what relationship groups are allowed? Friends? Family? Friends-of-Friends? And who can see what amongst what groups?
- You are required to think about this topic and write at least something up for iteration 1; depending on the domain it may be a little or a lot.

Iterative requirements capture

- You need to do several rounds on making the features/use-cases/UI sketches
- Use each one to help refine and expand the others
- You have two chances in your submissions, Iteration 1 and Iteration 2
- If we think your Iteration 1 is too incomplete or wrong we may ask you to revise it (for half the some points back) when you submit Iteration 2

Description of Use Cases

- Structured approach
  - Name
  - Short Description
  - Precondition: prerequisite for successful execution
  - Postcondition: system state after successful execution
  - Error situations: errors relevant to the problem domain
  - System state on the occurrence of an error
  - Actors that communicate with the use case
  - Trigger: events which initiate/start the use case
  - Standard process: individual steps to be taken
  - Alternative processes: deviations from the standard process

Use Case Description – Example

- Name: **Reserve Lecture Hall**
- Short description: An employee reserves a lecture hall at the university for an event.
- Precondition: The employee is authorized to reserve lecture halls
- Postcondition: A lecture hall is reserved
- Error situations: There is no free lecture hall
- System state in the event of an error: The employee has not reserved a lecture hall.
- Actors: **Employee**

Use Case Description – Example

- Trigger: Employee requires a lecture hall.
- Standard process:
  1. Employee logs in to the system.
  2. Employee selects the lecture hall.
  3. Employee selects the date.
  4. System confirms that the lecture hall is free.
  5. Employee confirms the reservation.
- Alternative processes:
  (4’) Lecture hall is not free
  (5’) System proposes an alternative lecture hall.
  (6’) Employee selects alternative lecture hall and confirms the reservation

Are there any questions?

ROAR