Requirements

Content adapted from CSE403 14sp by Michael Ernst

Outline

- What are requirements?
- How can we gather requirements?
- How can we document requirements?
- Use cases

Software Requirements

Requirements specify what to build:

- tell “what” and not “how”
- tell the problem, not the solution
- reflect system design, not software design

“what vs. how”: it’s relative

One person’s what is another person’s how.
– “One person’s constant is another person’s variable.” [Perlis]

- Input file processing is the what, parsing is the how
- Parsing is the what, a stack is the how
- A stack is the what, an array or a linked list is the how
- A linked list is the what, a doubly linked list is the how
- A doubly linked list is the what, Node* is the how

Why Requirements?

Some goals of doing requirements:

- understand precisely what is required of the software
- communicate this understanding precisely to all development parties
- control production to ensure that system meets specs (including changes)

Roles of Requirements

- customers:
  show what should be delivered; contractual base
- managers:
  a scheduling / progress indicator
- designers:
  provide a spec to design
- coders:
  list a range of acceptable implementations / output
- QA / testers:
  a basis for testing, validation, verification

Classifying requirements

The classic way to classify requirements:

- functional: map inputs to outputs
- nonfunctional: other constraints

Another way to classify them (S. Faulk, U. of Oregon):

- Behavioral (user-visible): about the artifact (often measurable)
- Development quality attributes: about the process (can be subjective)

Gather requirements from users

The #1 reason that projects succeed is user involvement

– Standish group survey of over 8000 projects

Why Working with Customer?

- Improves perceived development speed
- They don’t always know what they want
- They do know what they want, and it changes over time

"Digging" for requirements

Do:

- Talk to the users, or work with them, to learn how they work.
- Ask questions throughout the process to "dig" for requirements.
- Think about why users do something in your app, not just what.
- Allow (and expect) requirements to change later.

"Digging" for requirements

Don't:

- Describe complex business logic or rules of the system.
- Be too specific or detailed.
- Describe the exact user interface used to implement a feature.
- Try to think of everything ahead of time. (You will fail.)
- Add unnecessary features not wanted by the customers.

Feature Creep

feature creep: Gradual accumulation of features over time.

– Often has a negative overall effect on a large software project.

Good or bad requirements? (and why?)

- The system will enforce 6.5% sales tax on Washington purchases.
- The system shall display the elapsed time for the car to make one circuit around the track within 5 seconds, in hh:mm:ss format.
- The product will never crash. It will also be secure against hacks.

Good or bad requirements? (and why?)

- The server backend will be written using PHP or Ruby on Rails.
- The system will support a large number of connections at once, and each user will not experience slowness or lag.
- The user can choose a document type from the drop-down list.

How do we specify requirements?

- Prototype
- Use Cases
- Feature List
- Paper UI prototype

Use cases

- A use case is an example behavior of the system
- A use case characterizes a way of using a system
- It represents a dialog between a user and the system, from the user's point of view
- It captures functional requirements

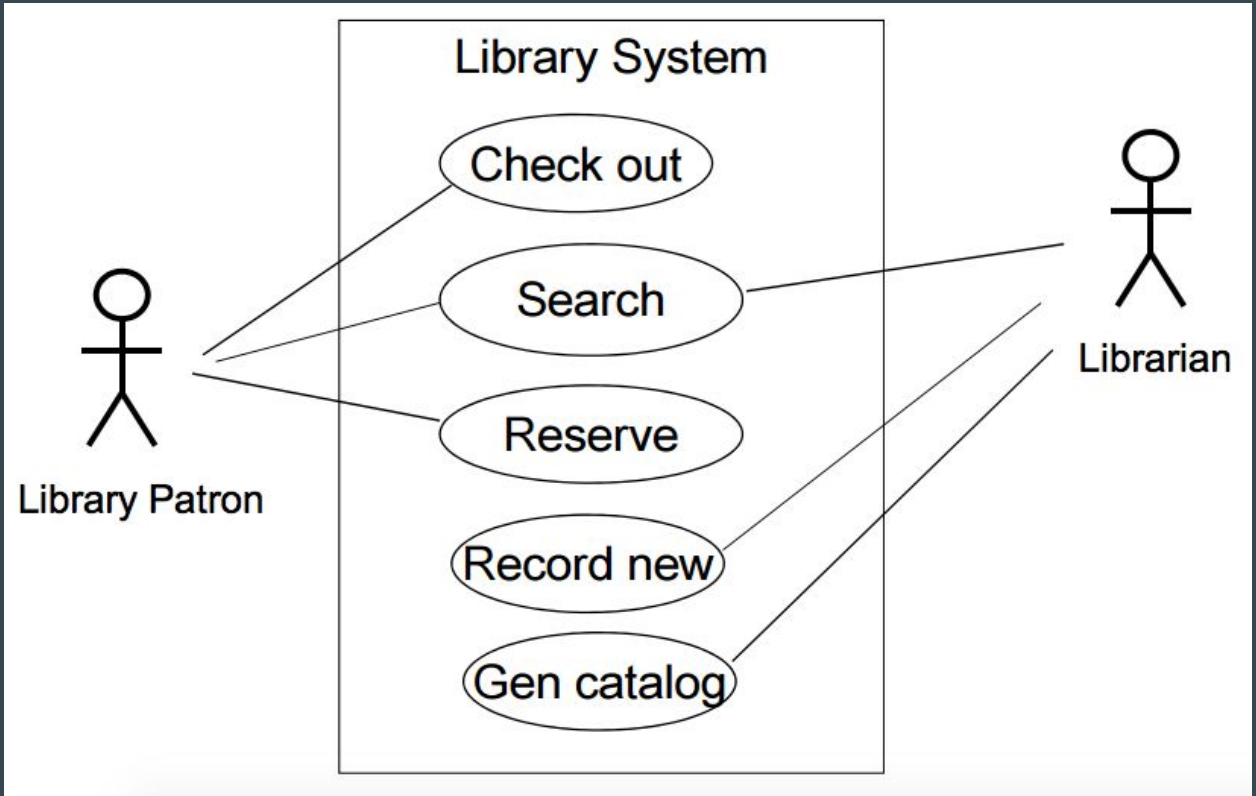
Qualities of a good use case

- starts with a request from an actor to the system
- ends with the production of all the answers to the request
- defines the interactions (between system and actors) related to the function
- takes into account the actor's point of view, not the system's

Qualities of a good use case

- focuses on interaction, not internal system activities
- NO GUI in detail
- has 3-9 steps in the main success scenario
- is easy to read
- summary fits on a page

Informal Use Case



Formal Use Case

Goal	Patron wishes to reserve a book using the online catalog
Primary actor	Patron
Scope	Library system
Level	User
Precondition	Patron is at the login screen
Success end condition	Book is reserved
Failure end condition	Book is not reserved
Trigger	Patron logs into system

Formal Use Case

Main Success Scenario	1. Patron enters account and password 2. System verifies and logs patron in 3. System presents catalog with search screen 4. Patron enters book title 5. System finds match and presents location choices to patron 6. Patron selects location and reserves book 7. System confirms reservation and re-presents catalog
Extensions (error scenarios)	2a. Password is incorrect 2a.1 System returns patron to login screen 2a.2 Patron backs out or tries again 5a. System cannot find book 5a.1 …
Variations (alternative scenarios)	4. Patron enters author or subject