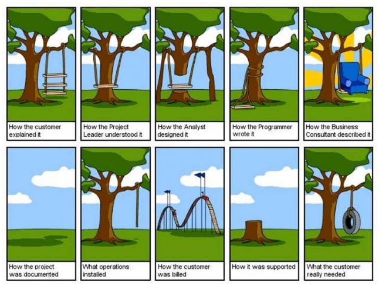
Requirements


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Lecture outline

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

Drumroll please …

- MusicLessonPlanner
- UMass Book Share

- SaveYour
- ArtRank

- BooleYou
- Shoptimize

Projects

- Check your email for your team assignment
- You guys made my job hard:
  - no clear consensus on top 6 projects
  - several people voted for 5 projects no one else liked
  - some people voted for all 5 projects everyone liked
- First assignment is requirements specification
  - due Feb 17

No time to waste

- Meet with your team ASAP
- Get organized
  - create an email list
  - create a calendar, place to keep notes, ideas
  - set a schedule
    - you must meet with TA for 30 min before Feb 17
    - you must meet with partner team before Feb 17
    - you must finish assignment by Feb 17
- Brainstorm ideas, features, product functionality

Notes

- We honored all reciprocal friends.
- Survey submission quality poor…dissapointing
- The project belongs to the whole team.
  All members are created equal.
- The teams and the individuals must ensure everyone contributes.

Lecture outline

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

Software requirements

- **requirements**: specify what to build
  - "what" and not "how"
  - the system design, not the software design
  - the problem, not the (detailed) solution

"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]

What	How
Parsing	Stack
Stack	Array or Linked List
Linked List	Doubly Linked List

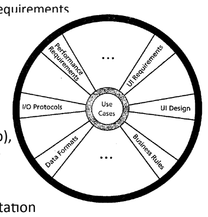
Why requirements?

- 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

Cockburn's requirements list

Requirements Outline: A template of all functional requirements

1. purpose and scope
2. terms / glossary
3. **use cases**
4. technology used
5. other
   - 5a. development process - participants, values (fast-good-cheap), visibility, competition, dependencies
   - 5b. business rules / constraints
   - 5c. performance demands
   - 5d. security (now a hot topic), documentation
   - 5e. usability
   - 5f. portability
   - 5g. unresolved / deferred
6. human issues: legal, political, organizational, training

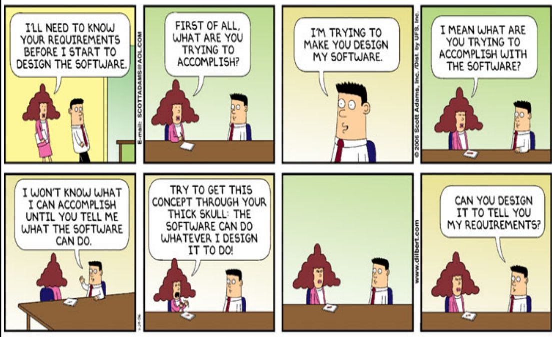


How do we gather requirements?

Let's start with two facts:

- Standish group survey of over 8000 projects, the number one reason that projects succeed is user involvement
- Easy access to end users is one of three critical success factors in rapid-development projects (McConnell)

Typical situation


I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?

How do we specify requirements?

- Prototype
- Use cases
- List of features
- Paper (UI) prototype

- System Requirements Specification Document

A good use case

- starts with a request from an actor to the system
- ends with the production of all answers to the request
- defines the interactions (between system and actors) related to the function
- from the actor's point of view, not the system's
- focuses on interaction, not internal system activities
- doesn't describe the GUI in detail
- has 3-9 steps in the main success scenario
- is easy to read
- summary fits on a page

Use cases

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.

Example:
Jane has a meeting at 10AM; when Jim tries to schedule another meeting for her at 10AM, he is notified about the conflict

Use case terminology

Actor: someone who interacts with the system

Primary actor: person who initiates the action

Goal: desired outcome of the primary actor

Level: top or implementation

What are some possible actors?

Do use cases capture these?

Which of these requirements should be represented directly in a use case?

1. Order cost = order item costs × 1.06 (tax)
2. Promotions may not run longer than 6 months.
3. Customers only become Preferred after 1 year
4. A customer has one and only one sales contact
5. Response time is less than 2 seconds
6. Uptime requirement is 99.8%
7. Number of simultaneous users will be 200 max

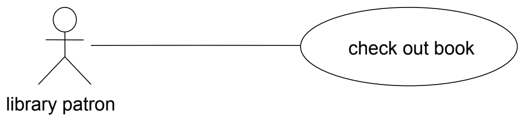
Three ways to write down use cases

- Diagrams
  - unified modeling language (UML)
- Informal language
- Formal specification

Use case summary diagrams

The overall list of your system's use cases can be drawn as high-level diagrams, with:

- actors as stick-men, with their names (nouns)
- use cases as ellipses with their names (verbs)
- line associations, connecting an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use / rely on

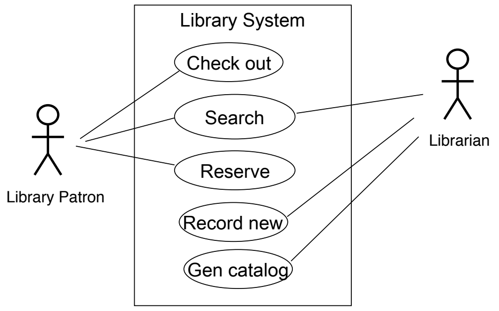


Use case summary diagrams

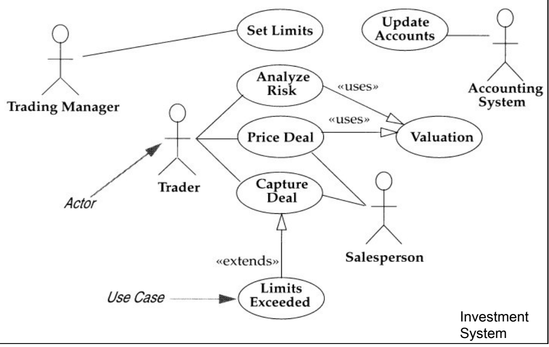
It can be useful to create a list or table of primary actors and their "goals"

Actor	Goal
Library Patron	Search for a book
	Check out a book
	Return a book
Librarian	Search for a book
	Check availability
	Request a book from another library

Use case summary diagram 1



Use case summary diagram 2



Informal use case

Informal use case is written as a paragraph describing the scenario/interaction

- Example:
  - Patron Loses a Book
    The library patron reports to the librarian that she has lost a book. The librarian prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The system will be updated to reflect lost book, and patron's record is updated as well. The head librarian may authorize purchase of a replacement tape.

Structured natural language

- I
  - I.A
    - I.A.ii
      - I.A.ii.3
        - » I.A.ii.3.q

Although not ideal, it is almost always better than unstructured natural language

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**

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**

Steps to creating a use case

- Identify actors and their goals
- Write the success scenario
  - identify happy path
- List the failure extensions
  - almost every step can fail
- List the variations
  - forks in the scenario

Jacobson example: recycling

The course of events starts when the customer presses the "Start-Button" on the customer panel. The panel's built-in sensors are thereby activated.

The customer can now return deposit items via the customer panel. The sensors inform the system that an object has been inserted, they also measure the deposit item and return the result to the system.

The system uses the measurement result to determine the type of deposit item: can, bottle or crate.

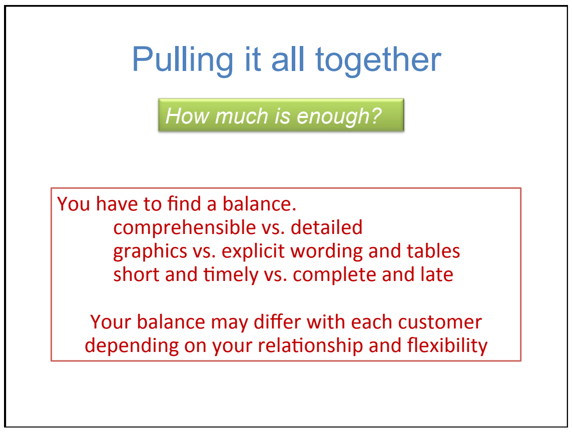
The day total for the received deposit item type is incremented as is the number of returned deposit items of the current type that this customer has returned…

Another example: buy a product

http://ontolog.cim3.net/cgi-bin/wiki.pl?UseCasesSimpleTextExample

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

- Alternative: **Authorization Failure**
  - At step 6, system fails to authorize credit purchase
  - Allow customer to re-enter credit card information and re-try
- Alternative: **Regular Customer**
  - 3a. System displays current shipping information, pricing information, and last four digits of credit card information
  - 3b. Customer may accept or override these defaults
  - Return to primary scenario at step 6



Pulling it all together

How much is enough?

You have to find a balance.
comprehensible vs. detailed
graphics vs. explicit wording and tables
short and timely vs. complete and late

Your balance may differ with each customer depending on your relationship and flexibility