CSE403 • Software engineering • sp12

Week 2				
Monday	Tuesday	Wednesday	Thursday	Friday
• ***Requirements*** • **Ambiguity** • **Don't write requirements in a bad mood** • **Why requirements?** • **Kinds of requirements** • **Use cases**	• Group meetings – let your group TA know where you meet	• Team work and structure	• SRS information	• Agile

A sign once outside the Blue Moon Tavern:
"We are not allowed to advertise that we sell beer here."

Ambiguity

Must I carry a dog?
What about the new shoes in my shopping bag?
Do dogs have to wear shoes?
What about an amputee? A single shoe? A double amputee?
What are shoes?
What are dogs?

On the street in front of a veterinarian's office: *"Please keep these parking spaces available for patients."*

Collect 'em, Share 'em with your friends!

Shoes Must Be Worn

Dogs Must Be Carried

Formalize

Must I carry a dog? No • What about the new shoes in my shopping bag? No • Do dogs have to wear shoes? Amputee? Double amputee? Yes • What are shoes, dogs?

$\forall x(\text{OnEscalator}(x) \rightarrow \exists y(\text{PairOfShoes}(y) \wedge \text{IsWearing}(x,y))$
$\forall x((\text{OnEscalator}(x) \wedge \text{IsDog}(x)) \rightarrow \text{IsCarried}(x)$

- Formalization does not by itself ensure unambiguous requirements
- The formalizations say "dogs are carried" and "shoes are worn" while the signs say "must be" – a difference in grammatical mood
  - *Indicative mood*: pertinent facts about the world • "Each seat is located in one and only one theater."
  - *Optative mood*: objectives of the system • "Better seats should be allocated before worse seats at the same price."
- Principle of uniform mood: Indicative and optative properties should be entirely separated in documents (a) to reduce author and reader confusion and (b) to help identify problems
- If the software works right, both sets of properties will hold as facts

Designations vs. definitions [M. Jackson]

- *Designations* are atomic phenomena – asserted, not proven or provable, connecting the domain to the system
  - `Mother(m,x)  // m is x's genetic mother`
- *Definitions* define terms using designations and other definitions
  - $\text{Child}(x,m) \equiv \text{Mother}(m,x)$
    $\text{Grandmother}(m,x) \equiv \exists y(\text{Mother}(m,y) \wedge \text{Mother}(y,x))$
- Use as few designations as possible
- Allows precision (even without formalism) and refutability
  - T/F? $\forall m,x\ (\text{Mother}(m,x) \rightarrow \neg \text{Mother}(x,m))$
  - T/F? $\forall x,y,m\ (\text{Mother}(m,x) \wedge \text{Mother}(m,y)) \rightarrow x=y$
  - May help understand if the designations/definitions make sense in the domain

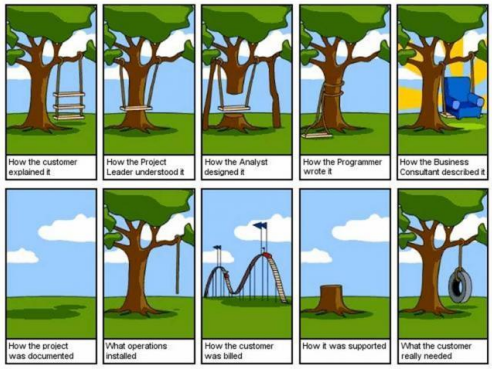
Requirements documents should

- avoid mood mixing
- use precise language – make sure to designate or define common terms that you are using with a specific meaning

Abstractions are key to requirements

- Y2K was (in a sense) a requirements problem
  - coders didn't consolidate date logic in one place
    - Another lesson from Y2K: "Premature optimization is the root of all evil." – Donald Knuth
  - should have had a requirement such as: *"The system must be easily modified to work in years ≥2000."*
- DRY principle: **Don't Repeat Yourself**
  - Abstractions live longer than details
  - A good abstraction allows appropriate change
  - But don't forget that ultimately your abstractions have to represent something useful in the domain

Why requirements?



Requirements: Goals and Roles

- Understand precisely what is required
- Communicate this understanding precisely to all parties
- Control production to ensure that system satisfies the (final) requirements

- Customers: show what should be delivered
- Managers: a scheduling and progress indicator
- Designers: provide a basis for design
- QA/testers: a basis for testing, validation, verification
- …

Good or bad requirements? Why?

Extra credit

- 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 five seconds, in hh:mm:ss format
- The product will never crash. It will also be secure against hacks
- 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

Classifying requirements

- functional: identify inputs and outputs of computation
  - "A user search is with respect to one or more of the defined databases."
  - "Every order is assigned a unique ID that the user can save."
- nonfunctional: other such as performance, dependability, reusability, safety, …
  - "Our deliverable documents shall conform to the XYZ process."
  - "The system shall not disclose any personal user information."

An alternative classification by Faulk in Reading II

"Digging" for requirements

Do

- Engage with the users to learn how they work
- Ask questions throughout the process
- Think about why users will do something in your system, not just what
- Allow and expect requirements to change later

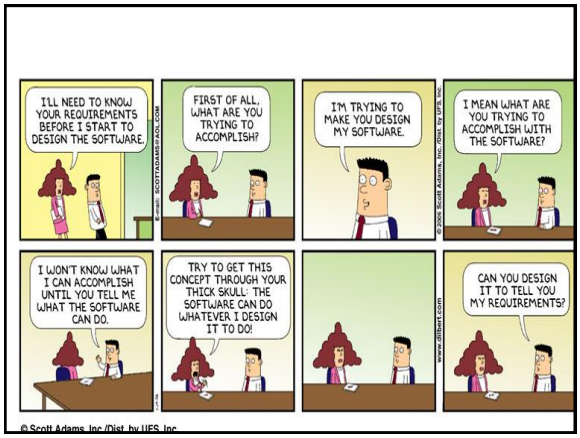
Don't

- Describe complex business logic or rules
- Be too specific or detailed
- Describe the exact user interface
- Try to think of everything ahead of time
- Add unnecessary features the customers don't want

Benefits of working with customers

- Good relations improve actual and perceived development speed
- Helps them figure out what they want
- Helps them change what they want more smoothly over time

- The #1 reason that projects succeed is user involvement [Standish Group]
- Easy access to end users is a critical success factor in rapid-development projects [McConnell]



"What" vs. "How"

- The conventional view is that requirements tell "what" to build and not "how" to build
- They reflect the problem, not the solution
- 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**

World and machine [Jackson]

Alternative to what vs. how

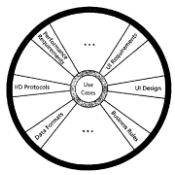
- The customer's requirements are in the application domain (the world)
- The program defines software (the machine) that has an effect in the world – for example, a database system dealing with books
- There are things in the world not represented by a given machine – for examples, book sequels or trilogies, pseudonyms, anonymous books, …
- Similarly, there are things in the machine that don't represent anything in the world – for example, null pointers, deleting a record, back pointers, …

How do we specify requirements?

- It varies, but usually some combination of
  - Prototype
  - Use Cases
  - Feature List
  - Paper UI prototype
- You will create a System Requirements Specification (SRS) document using structured natural language along with figures, use cases, etc.
  - I Although not ideal,
    - I.A structured natural language is almost always better
      - I.A.ii than unstructured natural language.
        - » I.A.ii.3 Unless the structure is an
          - » I.A.ii.3.q excuse to avoid content

Cockburn's requirements template

- purpose and scope
- terms / glossary
- **use cases (the central artifact of requirements)**
- technology used
- other
  - development process, participants, values (fast-good-cheap), visibility, competition, dependencies, business rules/constraints, performance demands, security, documentation, usability, portability, unresolved and deferred, human issues: legal, political, organizational, training



Use cases

- A *use case* is an example behavior of the system, representing specific flows of events in 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
  - Ex: *Robin has a meeting at 10AM; when Cameron tries to schedule another meeting for Robin at 10AM, Cameron is notified about the conflict*
- Similar to CRC (class responsibility collaborator) and eXtreme programming "stories"

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
9. Alternative: Authorization Failure
    1. At step 6, system fails to authorize credit purchase
    2. Allow customer to re-enter credit card information and re-try
10. Alternative: Regular Customer
    1. 3a. System displays current shipping information, pricing information, and last four digits of credit card information
    2. 3b. Customer may accept or override these defaults
    3. Return to primary scenario at step 6

Qualities of 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
- 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

Benefits of use cases

- Establish an understanding between the customer and the system developers of the requirements (success scenarios)
- Alert developers to problematic situations (extension scenarios)
- Capture a level of functionality to plan around (list of goals)

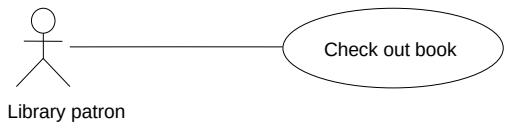
Terminology

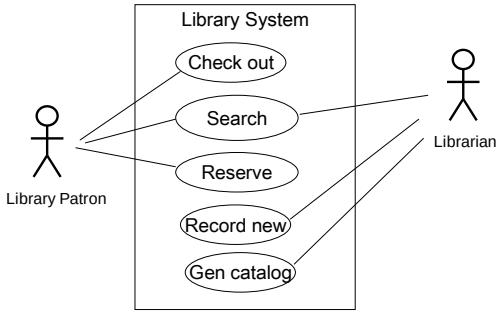
- Actor: someone/something who/that interacts with a use case; it could be a human, external hardware (like a timer), or another system
- Primary actor: actor initiating the action
- Goal: desired outcome of the primary actor
- Level: top-level or implementation
  - summary goals
  - user goals
  - subfunctions

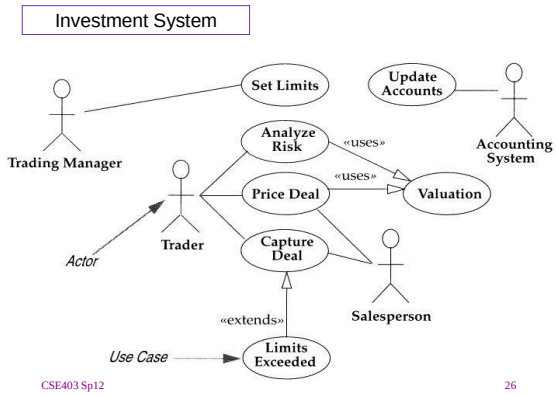
Use case summary diagrams

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

- actors as stick-figures with names (nouns)
- use cases as ovals with names (verbs)
- line associations that connect an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use







Are use cases good for these?

- Which of these requirements should be represented directly in a use case?
  - Order cost = order item costs * 1.06 tax.
  - Promotions may not run longer than 6 months.
  - Customers only become Preferred after 1 year
  - A customer has one and only one sales contact
  - Response time is less than 2 seconds
  - Uptime requirement is 99.8%
  - Number of simultaneous users will be 200 max
- None – many are non-functional requirements, others are core computation not based on interaction
  - Maybe the promotions, preferred, and sales contact would be handled in part with a use case

Use case summary diagrams

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

It can be useful to list the primary actors and their "goals" – the use cases they start

Informal use case

- An alternative, often combined with diagrams, is an informal use case written as a paragraph describing the interaction
- Ex: **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 book.

Formal use case: another approach

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

Creating a use case

- **Identify actors and their goals**
  - What computers, subsystems and people will drive our system? (actors)
  - What does each actor need our system to do? (goals)
- Consider software for a video store kiosk with no clerk
  - A customer with an account can use their membership and credit card to check out a video
  - The software can look up movies and actors by keywords
  - A customer can check out up to 3 movies, for 5 days each
  - Late fees can be paid at the time of return or at next checkout
- Exercises: (1) Name four use cases, and draw a UML-like use case summary diagram of the cases and their actors; (2) Write a formal use case for the Customer Checks Out a Movie scenario

Creating a use case

- **Write the success scenario**
  - This preferred "happy path" is easiest to read and understand, with everything else is a complication on this
  - Capture each actor's intent and responsibility, from trigger to goal delivery – say what information passes between them and number each line
- **List the variations**
  - Label variations with step number and alternative
    - 5'. Alternative 1 for step 5
    - 5''. Alternative 2 for step 5

What notation is good?

- There are standard templates for requirements documents, diagrams, etc. with specific rules. Is this a good thing? Should we use these standards or make up our own?
  - Good: standards are helpful as a template or starting point; others are more likely to understand
  - But don't be a slave to formal rules or use a model/scheme that doesn't fit your project's needs

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

Feature creep: a warning!

- Gradual accumulation of features over time
  - Often has a negative overall effect on a large software project
- Why does feature creep happen? Why is it bad?
- Because features are "fun"
  - developers like to code them
  - marketers like to brag about them
  - users want them
  - ... but too many – "it's OK, just one more" – means more bugs, more delays, less testing, ...

Week 2				
Monday	Tuesday	Wednesday	Thursday	Friday
• ***Requirements*** • …	• Group meetings – let your group TA know where you meet	• Team work and structure	• SRS information	• Agile

- Announcements now only on GoPost Announcements section
- Reading I: due tonight @ 11PM on DropBox
  https://catalyst.uw.edu/collectit/dropbox/notkin/20734
- Weekly team summary: due Friday @ 11PM on DropBox by each PM
- SRS: due Tuesday April 10 @ 11PM on DropBox by each PM

Any questions?