

Stakeholders and Levels

- Stakeholders
 - identify for each use case
 - identify goals
 - why do people want to do this?
- Levels
 - sea level
 - fish level
 - part of a bigger use case
 - kite level
 - summary use case
 - cloud level
 - very high summary
 - clam level
 - too detailed

497-2-11

The Writing Process

1. Identify system scope
2. Brainstorm actors
3. Brainstorm user goals
4. Write outermost summary use case
5. Improve summary use case
6. Add other use cases
 1. Write main success scenario
 2. Brainstorm extension conditions
 3. Write extension steps
 4. Turn complex extensions into separate use cases
7. Merge small use cases, split large ones
8. Go through all of them again

497-2-12

Brainstorming / Mindmapping

- activities to come up with ideas

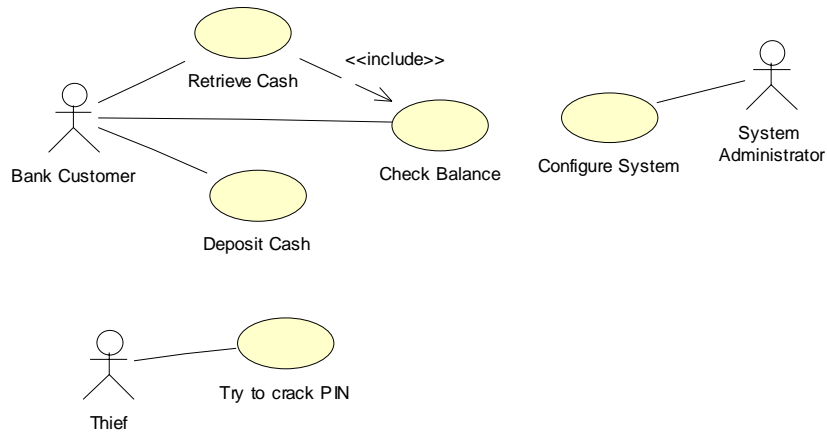
497-2-13

Mockups / Prototype

- A draft of the user interface
- Used to get feedback from users
- Talking to users in their terms
- May be realized as:
 - a running prototype (“rapid prototyping”)
 - a sketch on paper
- Prototype can also contain some functionality
- Tool: Gui builders
 - for example contained in MS Visual Studio
 - allow quick development of a running Gui that doesn’t do anything
 - uses drag-and-drop, requires almost no technical skill
 - limitations: non-standard Guis, dynamic elements

497-2-14

Use Case Diagram



497-2-15

Use Case Diagrams

- UML diagrams to show relationships between use cases and actors
- Actor
 - a role that a person (or machine) has when interacting with the program
- Use case relations
 - include: one use case includes another
 - other relations exist
 - but: most use cases are unrelated
- Really simple so that users don't get scared
 - no sequence, no control-flow
- Very imprecise (like most things in requirements)

497-2-16

Qualities of Requirements:

Unambiguity

- Clear, understandable
- For both you and your customer!
- Many computer science terms are ambiguous to others
 - example: "a or b"
 - in CS: can be: a and b
 - rest of the world (often): a or b, but not both
- Most English words have many different meanings
- Grammar can often be parsed in different ways
 - example (from Michael Jackson):

Signs at an escalator:
"Shoes must be worn"
"Dogs must be carried"

497-2-17

Qualities of Requirements:

Completeness

- You need to plan the project
 - hard if you don't know all the requirements
 - at least for the current increment
- Missing requirements...
 - may change the time plan
 - may change the architecture
 - may change risk
- Reasons for incompleteness
 - customers changing their minds
 - bad communication
 - forgot features needed by other features
 - example: user administration

497-2-18

Quality of Requirements:

Consistency

- Consistency
 - no contradictions
 - the program won't work if requirements are contradictory
- Hard to guarantee
 - large sets of requirements
 - contradictions can be hidden
 - example:
 - Section 1.2 says: "no more than 128MB of memory needed"
 - Section 5.8.9 says: "images should be rendered in real time"
 - => is this a contradiction???
- Formal logic: anything follows from a contradiction
- Problem:
 - often difficult to explain contradiction to customers
 - customers may want impossible things

497-2-19

Quality of Requirements:

Manageability

- Resources must match requirements
 - can it be done in time?
 - with the money available?
 - with the skills we have?
- Risk management
 - requirements should be prioritized
 - ideally: alternatives in case it doesn't work out
 - often impossible to tell which requirements are possible
- Complexity management
 - don't do everything at once
 - iterative processes
 - prototyping

497-2-20

Quality of Requirements:

Specificity

- Be as precise and detailed as possible
- Bad:
 - “program shall be fast”
 - “it takes a number as input”
- Good:
 - “the program shall give a response in less than 1s”
 - “it takes a 16-bit signed integer as input”
- Definitions
 - are often a good idea
 - example: “by 'Number', we always mean a 16-bit signed integer”
 - defined terms are often capitalized
- Rules about definitions
 - no circles

497-2-21

Quality of Requirements: No

Implementation Bias

- Implementation bias:
 - giving information about the design
 - giving information about the code
- Use terminology of the domain
 - not technical terminology
 - you want to focus on WHAT
 - leave HOW for later
- Bad examples:
 - “store the checked-out books in an array”
 - “calculate the square root with Newton's algorithm”
- Good examples:
 - “the library knows which books are checked out”
 - “return the square root with 5-digit precision”

497-2-22