

2. Requirements Capture

- First activity of the process
- Goal: find out what to build
- Communication between users and developers
 - thus: no complex notations possible (unless scientific domain)
 - mostly natural language
 - contract
- Methods for finding requirements
- Methods for formulating requirements
 - Scenarios, Use Cases, Mockups / Prototypes, Feature Lists
- Stakeholders
 - everybody interested in the product

497-2-1

Sources of Requirements:

Customers

- Interviewing customers
 - the people who pay you
 - other stakeholders
 - users
 - managers
- Problems:
 - customers may not know what they want / need
 - software is abstract and complex
 - may often change their minds
 - may not be able to express their needs in technical terms
 - communication between computer scientists / regular people
- Techniques
 - mockups, prototypes
 - walkthroughs
 - differences to existing systems

497-2-2

Sources of Requirements: Market

- Evaluating competing products
 - what has been done before?
 - where is a niche for us?
 - but take care not to violate copyrights, trademarks, patents
- Evaluating own abilities
 - what are we better at than the competition?
 - what knowledge, skills do we have?
- Market survey
 - interview consumers
 - difficult to do right; polling companies
 - marketing and advertising: create the demand
 - consider future trends
- Problems:
 - people don't know what they want
 - example: video telephones
 - markets change quickly

497-2-3

Sources of Requirements: Standards

- Standards and interoperating systems
 - system standards, file formats, network protocols
 - usability standards
 - domain standards
- Official standards
 - written by a standards body
 - ANSI
 - ISO (e.g. Unicode)
 - IEEE (e.g. Posix)
- Industry standards
 - Wintel Platform
 - Java, Dot-Net
 - Wimp user interface

497-2-4

Kinds of Requirements

- Functional
 - features
 - user interface
 - input / output
- Non-functional
 - user qualities
 - performance
 - precision
 - reliability
 - developer qualities
 - maintainability
 - reusability
 - interoperability

497-2-6

Scenarios and Use Cases

- Scenario
 - A sequence of events describing an interaction between the program and a user
- Use case
 - Set of related scenarios with a common goal

497-2-7

Scenarios

- Example:
 - Successfully retrieve cash from ATM**
 1. User enters ATM card
 2. ATM asks for PIN
 3. User enters PIN
 4. ATM displays menu
 5. User selects "Get cash - \$100"
 6. ATM disburses 5 bills for \$20
 7. User takes cash and ATM card
- Scenario captures a typical event sequence

497-2-8

Use Cases

- Related scenarios with common goal:
 - Successfully retrieve cash from ATM
 - Failed attempt to retrieve cash because ATM out of service
 - Failed attempt to retrieve cash because of insufficient funds
 - Failed attempt to retrieve cash because of invalid ATM card
 - Failed attempt to retrieve cash because of invalid PIN
 - ⇒ **grouped together as use case "Retrieve cash from ATM"**
- Elements:
 - Describe main success scenario (numbered steps)
 - List exceptions and variations separately
 - Short title that shows main goal, using imperative form
 - Start state, end state, additional conditions

497-2-9

Example of a Use Case

(abbreviated)

Retrieve cash from ATM

1. User enters ATM card
2. ATM asks for PIN
3. User enters PIN
4. ATM displays menu
5. User selects "Get cash - \$100"
6. ATM disburses 5 bills for \$20
7. User takes cash and ATM card

Variations:

- 1a. ATM says card is invalid or ATM out of service, end.
- 3a. ATM says PIN is invalid, repeat 3.
5. User may select \$20, \$40, or \$100. All amounts will be disbursed in \$20-bills.
- 5a. ATM says user's account has insufficient funds, end.

497-2-10

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

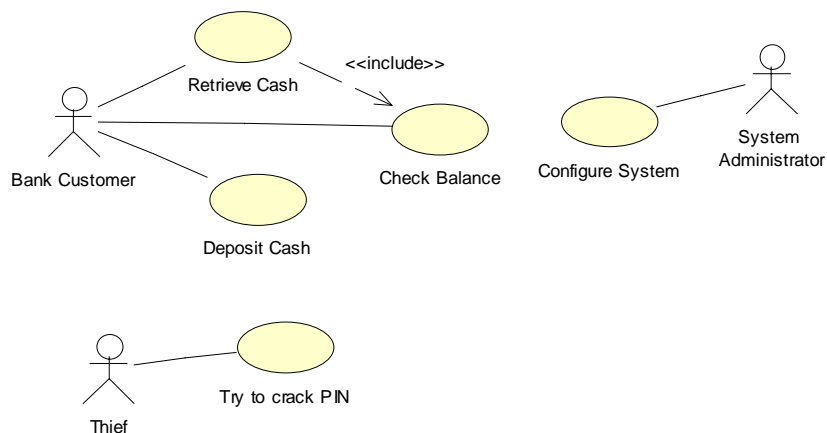
497-2-13

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

Nonfunctional Requirements

- Software Qualities
- External versus internal
 - external = for the user
 - internal = for the developer
- Product quality versus process quality
- Advice
 - be specific
 - prioritize
 - which are relevant for your project?
 - how to verify them?

497-2-23

Software Qualities

- Correctness
 - mathematical property
 - can be proven
 - in theory
 - based on specification
- Reliability
 - statistical
 - depends on operational profile
 - seriousness of error considered
- Robustness
 - failing gracefully
 - appropriate error messages
 - error recovery

497-2-24

Software Qualities

- Performance
 - time
 - space
 - algorithmic complexity
 - depends on hardware, OS,...
- Usability
 - easy to use
 - learnability
 - different for different users
 - newbies
 - experienced users

497-2-25

Software Qualities

- Verifiability
 - can be tested
 - can be understood
- Maintainability
 - repairability
 - evolvability
- Reusability
 - procedures / functions
 - libraries
 - components
- Portability
- Interoperability

497-2-26

Specialized Qualities: Distributed Systems

- Interoperability in the network
 - network protocols
 - platforms, users
- Scalability
 - works with many users
- Response time
 - similar to performance
 - perceived performance vs. actual performance
 - example: HTTP
- Distribution
 - client-server: centralized
 - no single point of failure
 - peer-to-peer

497-2-27