Requirements elicitation: Finding the Voice of the Customer

Establishing customer requirements for a software system

- Identify sources of user requirements on your project
- Identify different classes of users
- Gain access to representatives of those classes
- Agree on ultimate decision maker to resolve conflicting views

⇒Need iterative process to achieve shared understanding with users

Goal: Software Requirements Specification

- Introduction
  - Product scope, need for the system and fit with business objectives
- Overall description
  - high level functionality, users, environment, assumptions and dependencies
- External interface requirements
  - user, hardware, software, communications interfaces
- Functional requirements, system features
  - for each system feature: description, priority, etc
- Non-functional requirements
  - performance, safety, security, quality, business rules, user documentation
- Appendices: Glossary, Analysis models, TBD's

What is involved in requirements analysis and specification

- ***Precisely*** specifying the computing needs of an individual or an organization
- The specification needs to be in a "document" to ensure consistency, completeness and persistence
- Informal sections should be understandable to all who will be affected by the "system"
  - How it will affect their work
  - Allows them to contribute and become "satisfied customers"
  - In domain (user) language
- Formal sections: precise enough to form a contract

Stakeholders

People and organizations

- Who will be affected by the system
- Who have direct or indirect influence on the system requirements

Sources of requirements

- Interviews and discussions with potential users
- Documents that describe competing products
- System requirements specification
- *Problem reports and enhancement requests*
- *Market research*
- *Observing users*
- *Scenario analysis* of user tasks

User classes

How users differ:

- Their business domain or application,
- What they use the product for, the business processes they perform
- The frequency with which they use the product, their computer expertise
- Geographic location
- Access privileges

n.b. Users do not need to be human

Finding user representatives

- Group users appropriately
- Diversity
- Representative
- How to get user input: direct involvement in project, focus groups, interviews, surveys
- Possible connections between users and developers
  Figure 7.1 (Wie)

Product champion

- Wiegers uses terminology in a non-standard way
  - Schach more “standard” use of terminology
- User who acts as primary interface with developers
- Collect requirements from other members of their class

$\Rightarrow$ works with analyst to develop a unified set of requirements for their user class

Product champion activities

- Planning
- Requirements
- Verification and validation
- User aids
- Change management

Who makes decisions ??????????????????

Vision and scope should guide the discussions

- Product champions
- Most important user classes
- Major customers
- Avoid being an arbitrator
- The customer always has a point! (but may not be right!)

Misunderstanding who makes decisions is suicide
(Political awareness of the players in decisions too!)

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants
  - "wants" versus "needs" - what is the difference?
  - what about "remote" customers
    - who are they, what do they want, *what are their requirements*?
      - no real contract here, or is there?
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error
- Prototyping (discussed later) is an important technique of requirements validation

Hearing the *Voice of the Customer*

- Develop a set of detailed descriptions of how the system will be used -- User Scenarios
- Control complexity by grouping the users/tasks into *Use Cases* that describe units of functionality that have value
- Document your interviews, understand thought processes, and underlying logic -- refine, refine, refine
  - How will you structure your questions, the process?
    - Ref Gause & Weinberg

Hearing the *Voice of the Customer*: User Scenarios

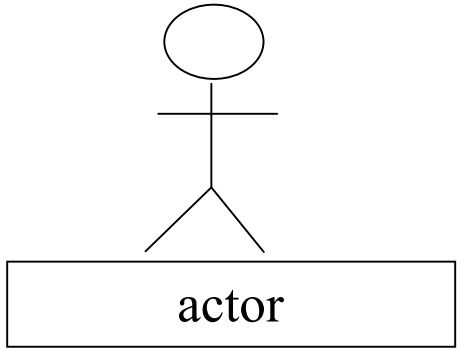
- A *User Scenario* is a concrete set of actions describing how a user will achieve a particular goal
- User Scenarios
  - Identify the different types of users
  - Observe users and develop a detailed description of what functionality the system provide them
  - Develop concrete examples of the future system in use
- Developing scenarios:
  - open ended questions to understand users current process
  - inquire about *exceptions and difficulties*
  - what would you need to know to do their job successfully

Documenting user requirements: *Use Cases*

- Use Cases
  - A set of abstract descriptions capturing all the behaviors of the system
  - *Defines the systems boundaries*
- Refine for *errors and exceptions*
- *Validate*

Use Case Diagrams: Actors

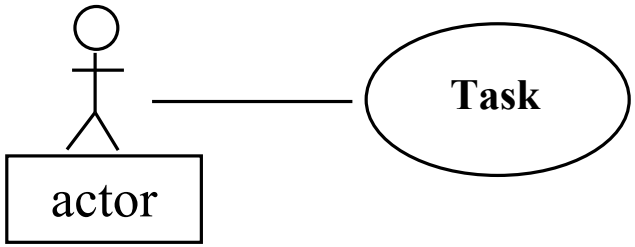
- A use case describes a sequence of interactions between a system and an external “actor” that results in the actor accomplishing a task that results in a benefit



- Person
- Software system
- Hardware system

Use Case Diagrams are about system function but begin with actors

- Actor = a role that a user plays with respect to the system
  - not a 1-1 relationship with actual people or things (other systems or system components) or organizations
  - carry out use cases
- Typically easier to determine actors before use cases
- Use cases are about system functionality, actors allow you to track who uses what functionality



Example: A small regional airline wants a system for scheduling flights and making reservations

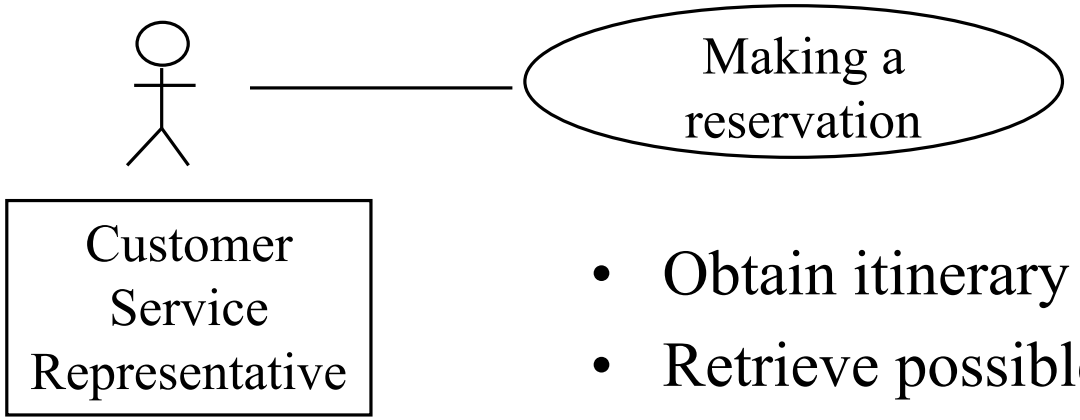
Actors:

- Administrator to set up and modify the airlines flight schedule
- Customer service representative
- Automated food ordering system (not a person!)

Example: A small regional airline wants a system for scheduling flights and making reservations

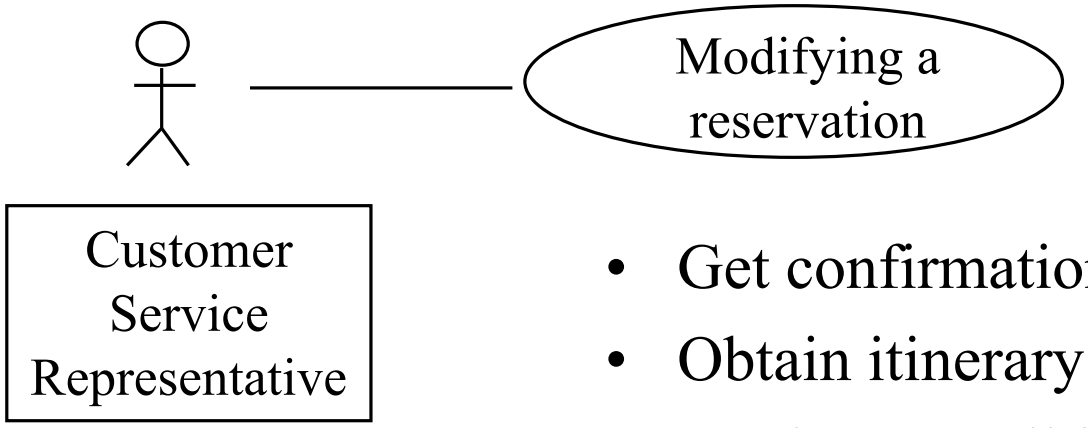
- Administrator to set up and modify the airlines flight schedule
  - enter airports
  - flight descriptions
  - scheduled flights
- Customer service representative
  - making/changing reservations
  - deleting reservations
- Automated food ordering

Use Case Diagrams: Making/modifying a reservation, normal course



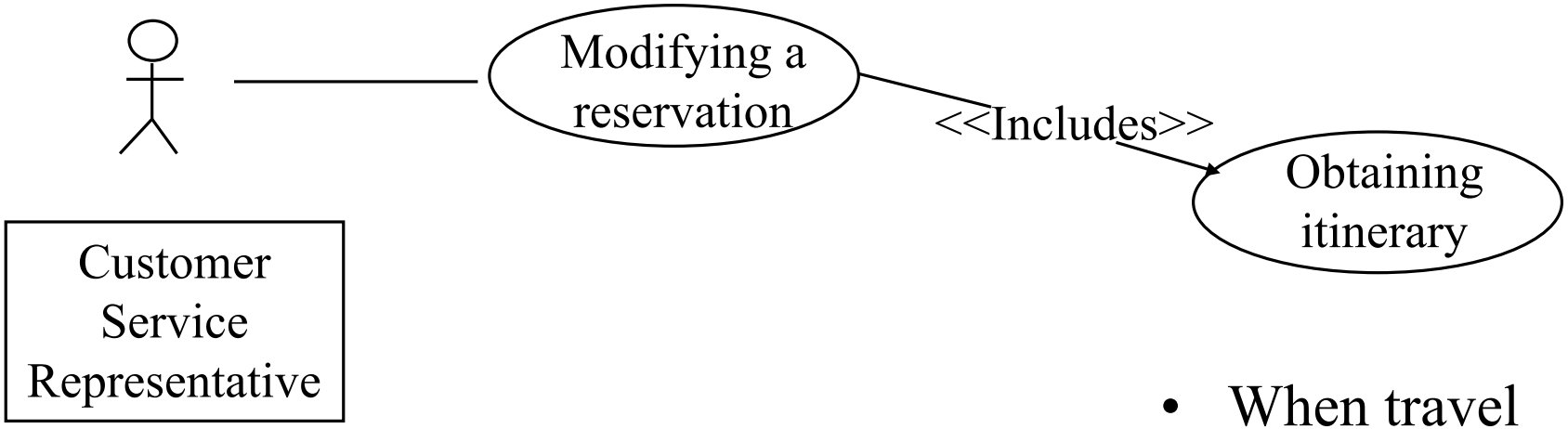
- Obtain itinerary
- Retrieve possible flights
- Enter reservation
- Seating assignment
- Meals? Get special requests
- Confirm with customer, confirmation #

Use Case Diagrams: Making/modifying a reservation, alternative course



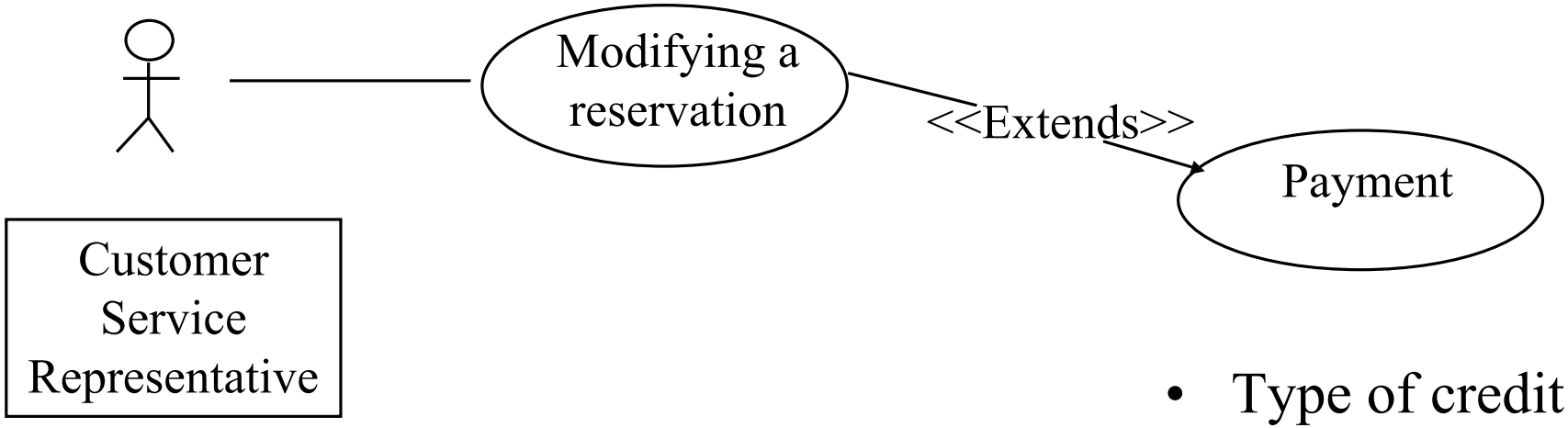
- Get confirmation number
- Obtain itinerary
- Retrieve possible flights
- Enter reservation
- Seating assignment
- Meals? Get special requests
- Confirm with customer, confirmation #

Modifying a reservation must include obtaining the itinerary <<include>>



- When travel
- Where travel
- # of travelers

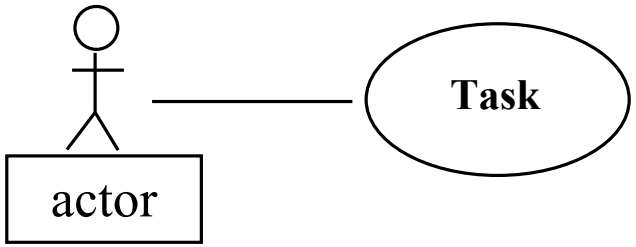
Modifying a reservation may include obtaining the method of payment <<extends>>



- Type of credit card
- Card #
- Expiration date
- …

Guidelines for Use Case Diagrams

- Avoid too many use cases
  - group logically related tasks, multiple scenarios
  - the primary scenario is the normal course of events
  - other scenarios are alternative courses
- Identify actors and roles first, then functionality or business processes
- focus on specific scenarios



Use Case documentation

- Use Case name: Making/modifying a reservation
- Participating Actor: Customer service representative
- Pre-condition: Appropriate flights are available
- Flow of events: …
- Post-condition: Reservation confirmed in system
- Special requirements: Ability to handle standard types of credit cards including …

Scenarios and Use Cases: Summary

- Identify types of users and User Scenarios
- Review with users for exceptions and special circumstances
- Use Cases (abstraction)
  - identify Actors
  - functionality achieves a discrete goal
- Use cases provide a mechanism to for documentation and refinement
- Review with users
- Validate

Requirements checking

- Validity: Does the system provide the functions which best support the customer's needs?
- Consistency: Are there *any requirements conflicts*?
- Completeness: Are all functions required by the customer included?
- Realism: Can the requirements be implemented given available budget and technology
  - exploratory (vertical) fast prototypes

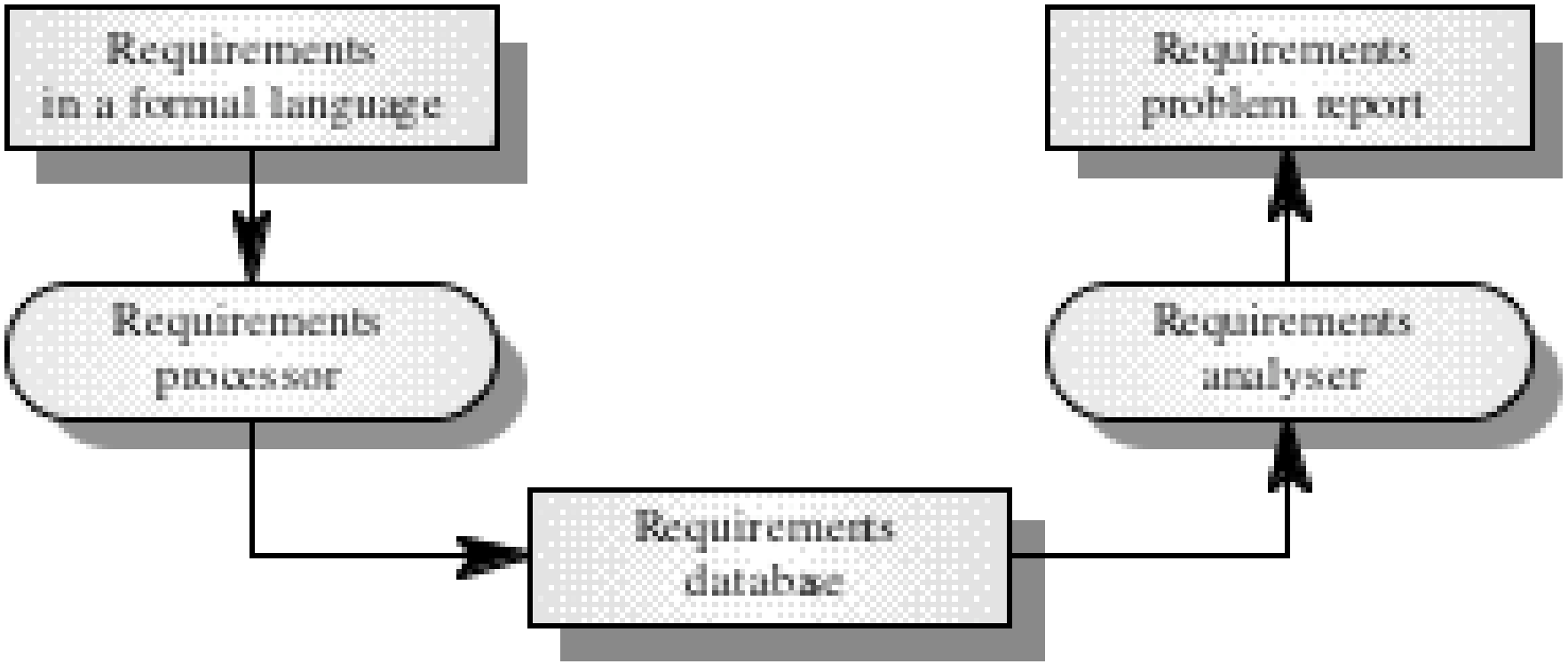
Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated
- Both client and contractor staff should be involved in reviews
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage

Review checks

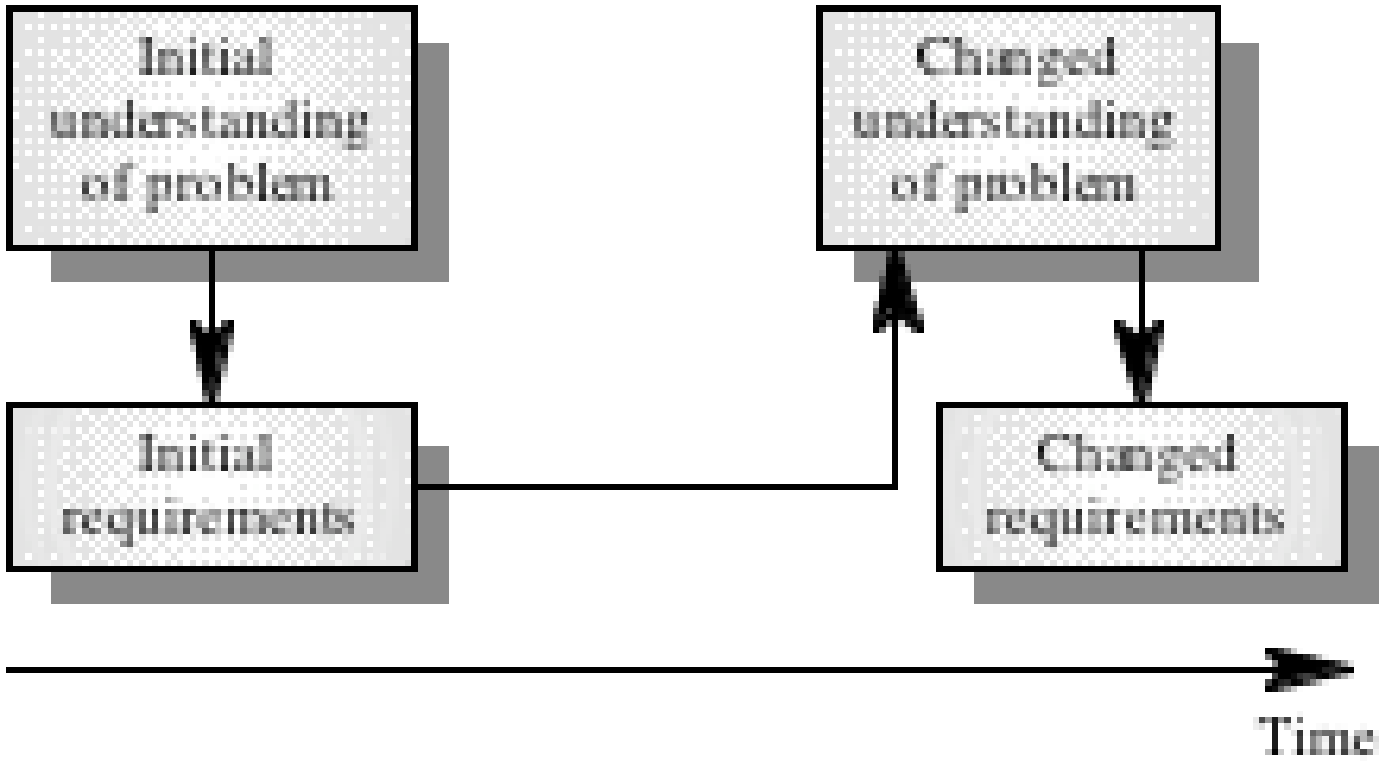
- Verifiability: Is the requirement realistically testable?
  - immediately imagine a simple test for each requirement
- Comprehensibility: Is the requirement properly understood?
  - multiple views
- Traceability: Is the origin of the requirement clearly stated?
  - boy is this important! - also record the "*priority*"
- Adaptability: Can the requirement be changed without a large impact on other requirements?

Automated consistency checking (RSL, Fisher)



Requirements evolution

- Requirements evolve as our understanding of user needs is developed and as the organization's objectives change
- It is essential to plan for change in the requirements as the system is being developed and used



Requirements definition/specification

- Requirements definition
  - A statement in natural language plus diagrams of the services the system provides and its operational constraints. Written for *customers*
- Requirements specification
  - A structured document setting out detailed descriptions of the system services. Written as a ***contract*** *between client and contractor*
- Software specification
  - A detailed software description which can serve as a basis for a design or implementation. Written for *developers*

Definitions and specifications

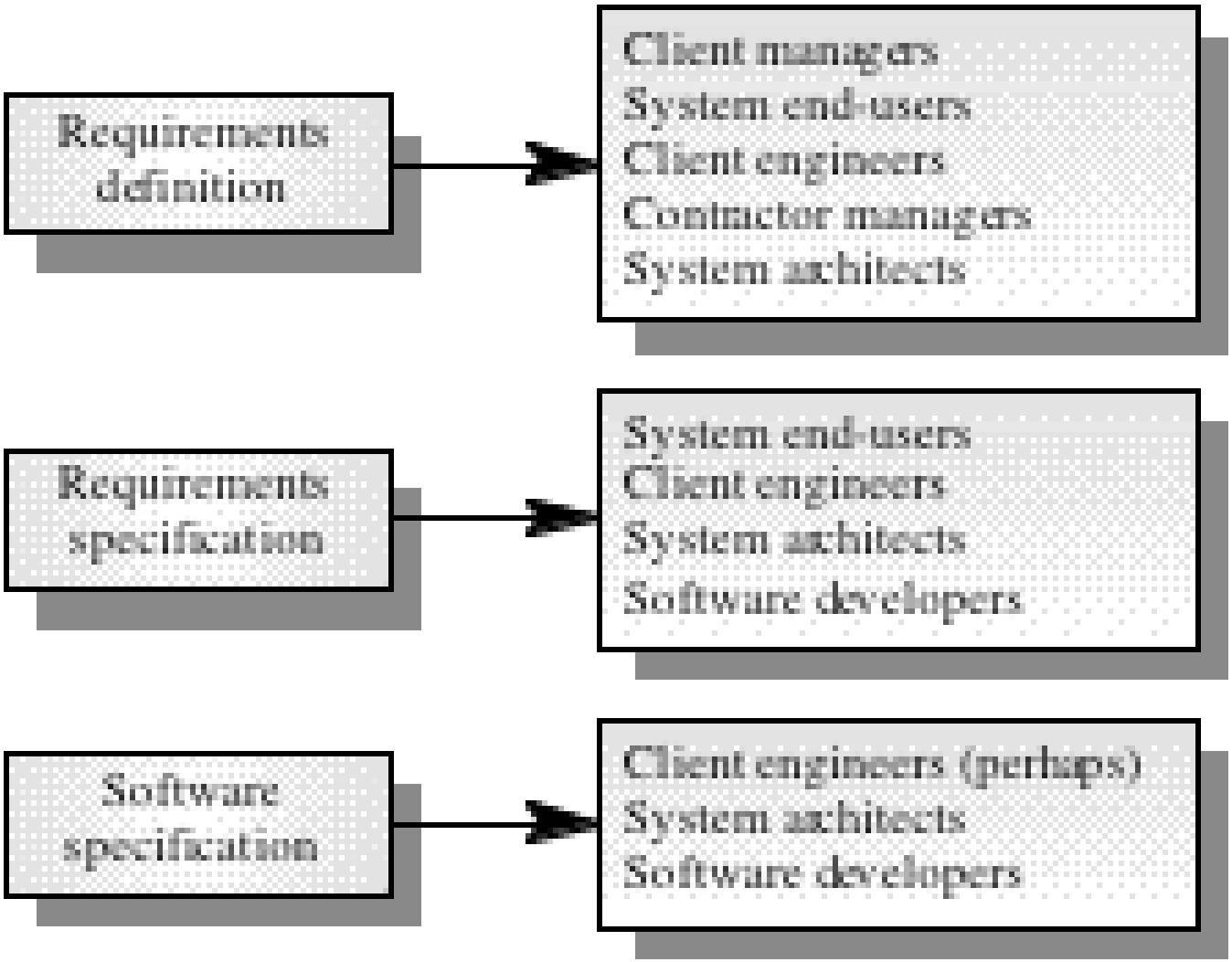
Requirements definition

1. The software must provide a means of representing and accessing external files created by other tools.

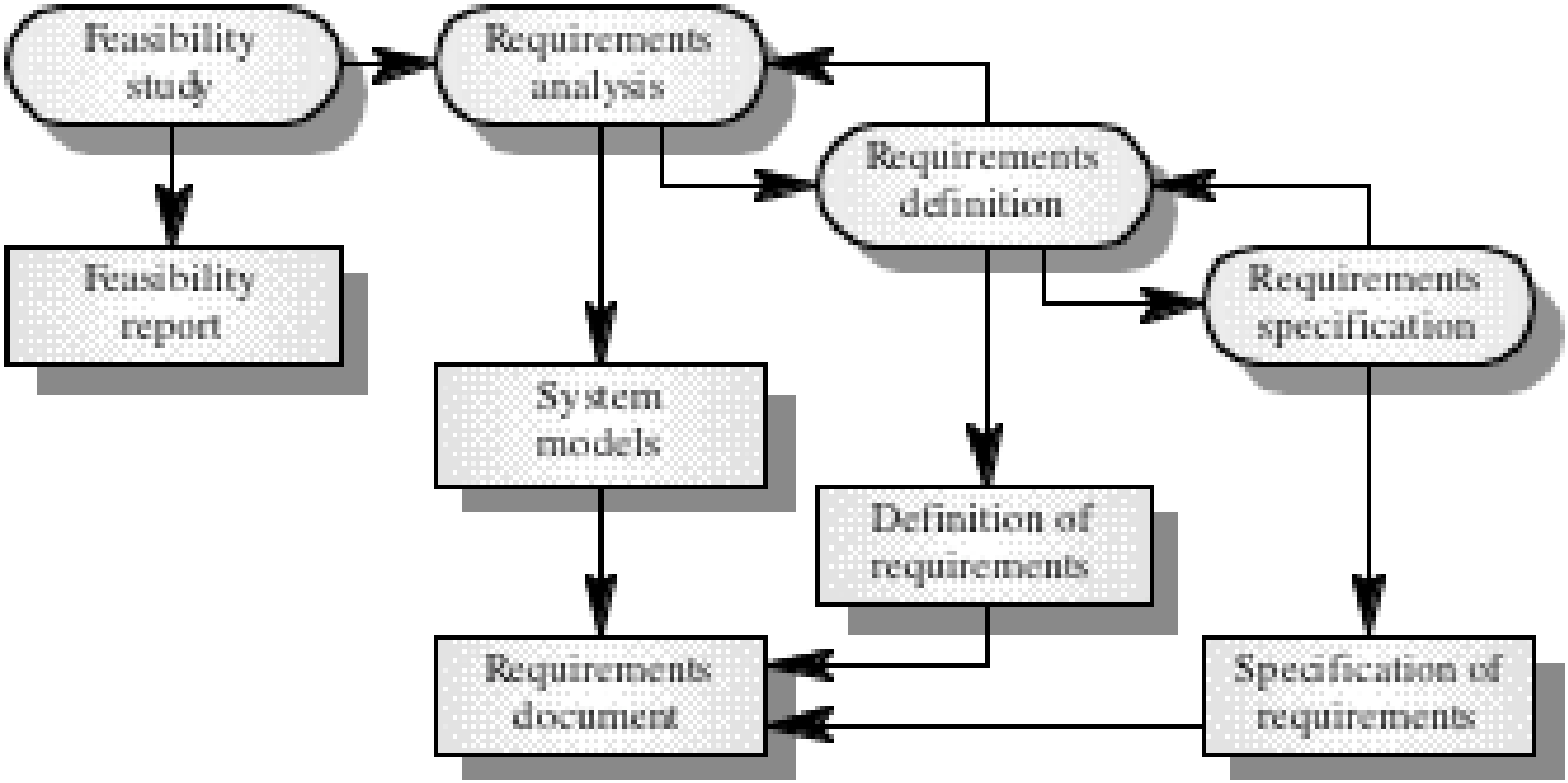
Requirements specification

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Requirements readers



The Requirements Engineering process



The requirements document

- The requirements document is the official statement of what is required of the system developers
- Should include both a definition and a specification of requirements
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it
  - This is important, keep the requirements document inside the "problem domain" not the "solution domain"
  - OK to record design ideas, but do it separately…

Requirements document requirements

- Specify external system behavior
  - user-visible behaviors
  - black box description
- Specify implementation constraints
  - what the system should NOT do
- Easy to change
- Serve as reference tool for maintenance
- Record forethought about the life cycle of the system i.e. predict changes
- Characterize appropriate responses to unexpected events

- Serve as reference for testing group
  - esp acceptance and system testing
- Serve as a basis for user documentation

Requirements for all Requirements Documentation

- Numbering consistently and meaningfully
- Use whitespace and page breaks!
- Use visual emphasis consistently and sparingly
- Always a TOC and Index
- Number all figures and tables, index them and refer to them by number
- Cross reference using tools (expect updates and changes!)
- Use TBD (and *have a list of them, dated, person resp.*)
- Careful when defining UI's.
  - conceptual drawings that do not create design expectations
- READ some example requirements documents (one by Wiegers, others provided on 308 webpage - don't slavishly copy structure but read for understanding of purpose, really!)
  - Different styles and structures can be fine if they support recording of the requirements.)

Requirements document structure

- Introduction
  - Describe need for the system and how it fits with business objectives
  - Document conventions
  - Intended audience and overall organization
  - Product scope (can ref the vision and scope document)
  - References

- Glossary
  - Define all technical terms used (and other terms that may cause confusion)
- Overall Description
  - Product Perspective (relation to other products, include its family)
  - Product functions at a high level
  - User Classes and Characteristics (priorities)
  - Operating Environment
  - Design and Implementation Constraints
    - anything that limits options
  - Assumptions and Dependencies

- External Interface Requirements : System Models
  - User Interfaces
    - GUI stds
    - Screen layout constraints
    - Standard buttons, functions or links that must appear
    - Shortcuts
    - Error message display standards
  - Hardware Interfaces
  - Software Interfaces
  - Communications Interfaces

- System Features
  - System feature X in short, simple statement
  - Description and Priority
  - Stimulus / Response Sequences
  - Functional Requirements
    - software capabilities that must be present to support the user to carry out services provided by the feature (or perform the task in the use case)

- Non-function-oriented requirements definition
  - Performance Requirements
    - like no. of users, throughput, load
    - quantify as much as possible
  - Safety Requirements
    - what the system can never do
    - support with cers, policies, regs and laws that affect it
  - Security Requirements
  - Software Quality Attributes
    - specific, quantitative and verifiable as possible (?!?)
  - Business rules (mandated roles)
  - User documentation required (user manual OR “help”?)

- Other Requirements
- System Evolution
  - Define fundamental assumptions on which the system is based and anticipated changes
- Appendices
  - Analysis models (if they are better done here)
  - TBD list
- Index

Guidelines for Writing Requirements

- No algorithm guarantees success
- Common sense
- Experience with problems
  - Failing and learning from it!
    - this really is a big deal
    - better at Cal Poly than in some other professional situation
      - at least for the major lessons :-)

Guidelines

- Keep sentences short and clear
- Use active voice (http://staff.jccc.net/pmcqueen/Tips/voice.htm)
- Use good grammar, spelling and punctuation
- Use terms consistently as defined in Glossary
- Use common requirements statement format
  - "The system shall..." followed by action verb, followed by an observable result (external, visible, black box)
- Avoid vague, ill defined terms like
  - user friendly, robust, fast, state-of-the-art, acceptable, etc.
    - clarify with the customer how to measure the qualities used
    - clarify if customer wants to "process" or "manage" or "support"

- Avoid comparative words (see “bad words” list!)
  - improved, maximal, optimal
    - quantify the degree of the quality desired to clarify
- Other general tips:
  - Finding the right level of granularity
    - write individually testable requirements
      - if you can think of a small number of related test cases, that is about right
    - write at consistent level of detail
    - don’t write long, compound sentences (avoid “and/or, etc at all cost!)
    - avoid redundancy (tradeoff between understandable, boring and x-ref)
    - use tabular format where it makes sense
      - if there are cases that can be broken along certain dimensions

Sample Requirements (Wie. p. 165 ...)

- “The product shall provide status messages at regular intervals not less than every 60 seconds.”
- “The product shall switch between displaying and hiding nonprinting characters instantaneously.”
- “The parser shall produce an HTML markup error report that allows quick resolution of errors when used by HTML novices.”
- Charge numbers should be validated online against the master corporate charge number list, if possible.”
- “The product shall not offer search and replace options that could have disastrous results.”

The Data Dictionary

- Primitive data elements
  - Request ID = *6 digit system-generated sequential integer, beginning with 1, that uniquely identifies each request*
- Composition

  Requested Chemical = Chemical ID
  + Quantity
  + Quantity Units
  + (Optional Vendor Name)

- Iteration

Request	=	Request ID
	+	Charge Number
	+	1:10{Requested Chemical}

- Selection
  - Quantity Units = ["grams" | "kilograms" | "each"]
    + *9 character text string indicating the units associated with the quantity of chemical requested*