Requirements

Customer has a “notion”:
  automate a new transaction / application
  enhance an existing system
  replace an existing system

Requirement:
  a feature of the system
  a capability
  a set of functions
Requirements Elicitation and Analysis

- Problem
- Problem analysis
- Problem description

Customer and user needs: Did we capture all needs? Are we using the right views and techniques?

Determine the requirements
Document them in written form
Use some appropriate notation (UML, SDL, pseudo code)

Requirements Definition and Specification

- Problem description
- Prototyping and testing
- Problem documentation and validation

Are we using the right views and techniques? Is this capability feasible and required? Have we captured what the user needs and expects?

Determine the requirements
Document them in written form
Use some appropriate notation (UML, SDL, pseudo code)
The Importance of Requirements

The Standish survey reports the top causes of failed projects as:
1. Incomplete requirements - 13.1%
2. Lack of user involvement - (12.4%)
3. Lack of resources - (10.6)
4. Unrealistic expectations - (9.9%)
5. Lack of executive support - (9.3%)
6. Changes in requirements and specs - (8.7%)
7. Lack of planning - (8.1%)
8. System no longer needed - (7.5%)

Requirements Documents

- **Requirements Definition:**
  - Complete listing of customers expectations
  - Complete listing of user needs
  - Understanding between customer and developer
  - Jointly written by customer and developer

Subsequently rewritten as

- **Requirements Specifications:**
  - Restate requirements in technical terms
  - Both aspects are required, possibly as one document
  - Direct correspondence between both documents
  - Leads to Configuration Management
Final Result?

Does this solution fulfill the customers and users requirements and expectations???

Categories of Requirements

- Requirements that absolutely must be met
  - essential
  - necessary, sufficient

- Requirements that are highly desirable but not essential
  - “nice to have”

- Requirements that are possible but could be eliminated
  - “add-on” feature
Functional and Nonfunctional Requirements

- **Functional Requirements** - describe interactions between the system and its environment:
  - How should the system react to stimuli
  - What states are acceptable
  - What input is required, what output produced
  - Which reaction time is acceptable
  - 
  "use cases"

- **Nonfunctional Requirements** - constraints
  - Restrictions on the system
  - Restrictions on the construction
  - 

Where do requirements come from?

- **user**
- **customer**
- **Domain Expert**
- "Programmer"
- **developer**

Feb. 12
Types of Requirements - 1

Users and human factors
- Who will use the system
- Will there be several types of users
- What is the skill level of each type of user
- What kind of training will be required
- How easy will it be to understand and use the system
- How difficult will it be for a user to ab-/misuse the system

Functionality
- What will the system do
- When will the system do it
- Are there several modes of operation
- How and when can the system be changed/enhanced
- Are there constraints on speed, response time, throughput

Types of Requirements - 2

Physical environment
- Where is the equipment to function
- Is there one location or are there several
- Are there any environmental restrictions

Interfaces
- Is the input coming from one or more other systems
- Is the output going to one or more other systems
- Is there a prescribed way in which data must be formatted
- Is there a prescribed medium that the data must use

Documentation
- How much documentation is required (type of ..)
- In which form is it required (on-line, manual, book format . .)
- For which audience is it intended (user, operator, maintainer . .)
Types of Requirements - 3

- **Data**
  - What format should be used for input/output/storage
  - How often will data be received/sent
  - What is the accuracy (10^-9/32-bit - real world/model world)
  - What is the desired precision of the calculations
  - How much data is flowing through the system
  - Must any data be retained for any period of time

- **Resources**
  - What materials, personnel, other resources are required to build, operate, use and maintain the system
  - What skills must the developers have
  - Space, power, air conditioning etc. requirements
  - Is there a prescribed timetable / schedule / time limit
  - What is the financial limit for hardware/software/development

Types of Requirements - 4

- **Quality Assurance**
  - Reliability, availability, maintainability, security requirements and possible other quality attributes
  - What are acceptance/demonstration of characteristics criteria
  - Must the system detect and isolate faults
  - What is the prescribed mean time between failures (MTBF)
  - Maximum time allowed for restarting the system after a failure
  - How can the system incorporate changes to the design
  - Will maintenance merely correct errors or will it include improving the system
  - What measures/metrics will apply to efficiency (resource usage, response time, . . .)
  - How easy should it be to move the system from one location to another or from one type of network to another
  - How easy should it be to port the system from one platform to another (hardware / software)
  - ?
Types of Requirements - 5

**Security**
- Must access to the system or to information be controlled
- Will the system be isolated or connected to a network / the Internet
- How will one user’s data be isolated from others
- How will user programs be isolated from other programs
- How will user programs be isolated from the operating system
- Must log files be kept – and for how long
- How often will the system be backed up
- ?

? Any other requirements

The Real and the Model World

- Entities
- Relationships
- Activities

- data items
- associations
- functions

transformation / mapping
Mapping

- Real world items to model world concepts
  - entity ---> data structure
  - activity ---> function
  - relationship --> association, aggregation, composition

- The different approaches:
  - Data-oriented – typically found in the commercial world
  - Function-oriented – typically found in the technical world
  - Object-oriented – attempt to limit the scope to those entities and functions which are tightly coupled

Configuration Management

- A set of procedures that track:
  - The requirements that define WHAT the system should do
  - The design modules that are generated from the requirements (HOW the system should do it)
  - The program code that implements the design
  - The tests that verify the functionality
  - The documents that describe the system

- Configuration management ties the phases and the system parts together.
Characteristics of Requirements

- Are the requirements correct?
- Are the requirements complete? (external/internal)
- Are the requirements consistent?
- Are the requirements realistic?
- Are all requirements needed?
- Are the requirements verifiable? (validate)
- Are the requirements traceable?
- Are the requirements quantified?
  (qualitative vs. quantitative requirements)

How to Express the Requirements 9/7

- Static descriptions - relational
  - Indirect reference
  - Recurrence relation
  - Axiomatic definition
  - Expression as a language – pseudocode
  - Data abstraction – type/class/object
- Dynamic descriptions - states, stimuli, transitions
  - Decision tables
  - Finite state machines
  - Functional descriptions and transition diagrams
  - Event tables
  - Petri Nets
How to Express the Requirements

☞ UML - Unified Modeling Language
  • Based on several OOA&D methods,
  • most notably on methods developed by Booch, Rumbaugh and by Jacobson
  • Standardized by OMG

☞ UML is a modeling language, not a method
  • Uses a predominantly graphical notation
  • Does not really describe a process
  • But is supported by the Rational Unified Process (RUP)

☞ Reading assignment: Read chapters 1 and 2 in Fowler’s book "UML Distilled"

Object-Oriented Specification

☞ Functional view: input-output transformation
  • HIPO – hierarchical input/output (historical)

☞ O-O view: entity - relationship functionality (s/d)
  • What are the basic entities
  • How do they interact
  • What data structures represent entities
  • How does an entity’s state evolve over time
  • How does a state change occur
  • What constitutes an entity – how is it composed
Object-Oriention

- A technique for systems modeling

- A system is composed of a number of entities which interact
  - Entities are seen as “objects”
  - State is represented by “attributes”
  - Operations are represented as “methods”
  - Only methods can change the state (value of attributes)

Real World and Model World

model world

- Person
- House
- Tom
- Car
- T_house
- T_car

real world

- Person
- House
- Car
- Tree
- Tom
- Ford
- Elm
Diagrams in the UML

- Use Case diagram
- Class diagram
- Object diagram
- Sequence diagram
- Collaboration/Communication diagram
- Statechart diagram
- Activity diagram
- Component diagram
- Deployment diagram

Use Case Diagrams

A use case diagram shows a set of use cases and actors and their relationships. (Actors are outside the system and can be considered a special kind of class). Use case diagrams address the static use case view of a system. These diagrams are especially important in organizing and modeling the behaviors of a system.
Class & Object Diagrams

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams address the static design view of a system. Class diagrams that include active classes address the static process view of a system.

An object diagram shows a set of objects (class instances) and their relationships. Object diagrams represent static snapshots of instances of the things found in class diagrams. These diagrams address the static design view or static process view of a system as do class diagrams, but from the perspective of real or prototypical cases.

Sequence & Collaboration Diagrams

Both sequence diagrams and collaboration diagrams are kinds of interaction diagrams documenting the dynamic behavior of a system.

An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Interaction diagrams address the dynamic view of a system.

A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages; a collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.

Sequence diagrams and collaboration diagrams are isomorphic, meaning that one can transform one into the other and vice versa.
Statechart & Activity Diagrams

A statechart diagram shows a state machine, consisting of states, transitions, events, and activities. Statechart diagrams address the dynamic view of a system. They are especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems.

An activity diagram is a special kind of a statechart diagram that shows the flow from activity to activity within a system. Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.

Component & Deployment Diagrams

A component diagram shows the organizations and dependencies among a set of components. Component diagrams address the static implementation view of a system. They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations.

A deployment diagram shows the configuration of runtime processing nodes and the components that live on them. Deployment diagrams address the static deployment view of an architecture. They are related to component diagrams in that a node typically encloses one or more components.