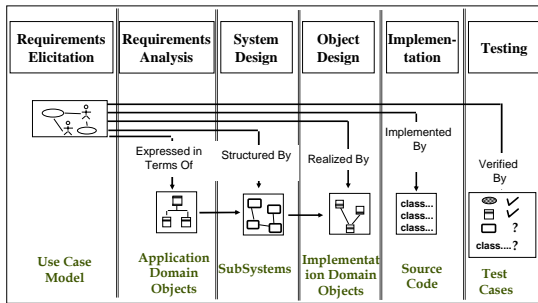
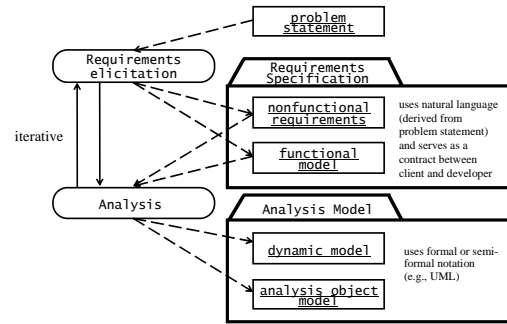


### Software Lifecycle Activities

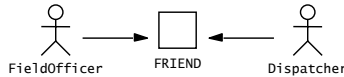


### Products of requirements elicitation and analysis



### Requirements Elicitation Activities

- Identify actors
- Identify scenarios
- Identify use cases
- Identify relationships among use cases
- Refine use cases
- Identify nonfunctional requirements
- Identify participating objects



### System Identification

- Development of a system is not just done by taking a snapshot of a scene (domain)
- Definition of the system boundary
  - What is inside, what is outside?
- How can we identify the purpose of a system?
- Requirements Process:
  - Requirements Elicitation: Definition of the system in terms understood by the customer
  - Analysis: Technical specification of the system in terms understood by the developer.

### Requirements Elicitation

- Challenging activity
- Requires collaboration of people with different backgrounds
  - User with application domain knowledge
  - Developer with implementation domain knowledge
- Bridging the gap between user and developer:
  - Scenarios: Example of the use of the system in terms of a series of interactions with between the user and the system
  - Use cases: Abstraction that describes a class of scenarios

### Types of Requirements

- Functional requirements:
  - Describe the interactions between the system and its environment independent from implementation (e.g., the watch system must display the time based on its location)
  - Focus on the purpose of the system rather than how the purpose is achieved or how efficiently the purpose is achieved

### *Types of Requirements*

- ◆ **Nonfunctional requirements:** User observable aspects of the system not directly related to purpose but often specified at the same time as functional requirements.
  - Look and feel of the user interface
  - Documentation format requirements
  - Hardware
  - Performance
    - ◆ The response time must be less than 1 second
    - ◆ The accuracy must be within a second
  - Error handling
  - Quality (reliability/availability)
    - ◆ The watch must be available 24 hours a day except from 2:00am-2:01am and 3:00am-3:01am
  - Security
  - Resources
    - ◆ Memory usage
    - ◆ Cpu speed

### *Types of Requirements*

- ◆ **Constraints** (“Pseudo requirements”): Imposed by the client or the environment in which the system will operate. Restricts the implementation of the system.
  - The implementation language must be Java.
  - Must interface to the dispatcher system written in 1956.
  - Must use Intel CPU

### *What is usually not in the Requirements?*

- ◆ System structure, implementation technology
- ◆ Development methodology
- ◆ Development environment
- ◆ Implementation language
- ◆ Reusability

It is desirable that none of the above are constrained by the client.

### *Requirements Validation*

- ◆ Critical step in the development process,
  - Usually after requirements elicitation or analysis. Also done at delivery.
- ◆ Requirements validation criteria:
  - **Correctness:**
    - ◆ The requirements represent the client’s view.
  - **Completeness:**
    - ◆ All possible scenarios through the system are described, including exceptional behavior by the user or the system
  - **Consistency:**
    - ◆ There are no functional or nonfunctional requirements that contradict each other.
  - **Clarity:**
    - ◆ There are no ambiguities in the requirements.

### *Requirements Validation Criteria (continued)*

- ◆ **Realism:**
  - Requirements can be implemented and delivered.
- ◆ **Traceability:**
  - Each system function can be traced to a corresponding set of functional requirements.

### *Types of Requirements Elicitation*

- ◆ **Greenfield Engineering**
  - Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client
  - Triggered by user needs
- ◆ **Re-engineering**
  - Re-design and/or re-implementation of an existing system using newer technology
  - Triggered by technology enabler
- ◆ **Interface Engineering**
  - Provide the services of an existing system in a new environment
  - Triggered by technology enabler or new market needs

### *Scenarios*

- ◆ “A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carrol, Scenario-based Design, Wiley, 1995].
- ◆ A concrete, focused, informal description of a single feature of the system used by a single actor.
- ◆ Scenarios can have many different uses during the software lifecycle.

### *Types of Scenarios*

- ◆ As-is scenario
  - ◆ Used in describing a current situation. Usually used during re-engineering. The user describes the system.
- ◆ Visionary scenario
  - ◆ Used to describe a future system. Usually described in greenfield engineering or reengineering.
  - ◆ Can often not be done by the user or developer alone
- ◆ Evaluation scenario
  - ◆ User tasks against which the system is to be evaluated
- ◆ Training scenario
  - ◆ Step by step instructions designed to guide a novice user through a system

### *How do we find scenarios?*

- ◆ Don't expect the client to be verbal if the system does not exist (greenfield engineering)
- ◆ Don't wait for information even if the system exists
- ◆ Engage in a dialectic approach (evolutionary, incremental)
  - ◆ You help the client to formulate the requirements
  - ◆ The client helps you to understand the requirements
  - ◆ The requirements evolve while the scenarios are being developed

### *Heuristics for finding Scenarios*

- ◆ Ask yourself or the client the following questions:
  - ◆ What are the primary tasks that the system needs to perform?
  - ◆ What data will the actor create, store, change, remove or add in the system?
  - ◆ What external changes does the system need to know about?
  - ◆ What changes or events will the actor of the system need to be informed about?
- ◆ Insist on task observation if the system already exists (interface engineering or reengineering)
  - ◆ Ask to speak to the end user, not just to the software contractor
  - ◆ Expect resistance and try to overcome it

### *Summary*

- ◆ Requirements Elicitation:
  - ◆ Greenfield Engineering, Reengineering, Interface Engineering
- ◆ Scenarios:
  - ◆ Great way to establish communication with client
  - ◆ As-Is Scenarios, Visionary scenarios, Evaluation scenarios Training scenarios
  - ◆ Use cases: Abstraction of scenarios
- ◆ Pure functional decomposition is bad:
  - ◆ Leads to un-maintainable code
- ◆ Pure object identification is bad:
  - ◆ May lead to wrong objects, wrong attributes, wrong methods
- ◆ The key to successful analysis:
  - ◆ Start with use cases and then find the participating objects
  - ◆ If somebody asks “What is this?”, do not answer right away. Return the question or observe: “What is it used for?”