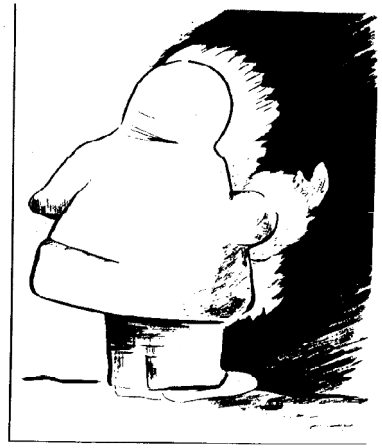
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Chapter 4, Requirements Elicitation

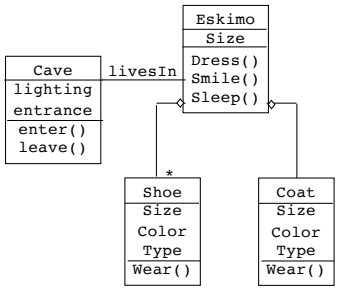
1

What is This?



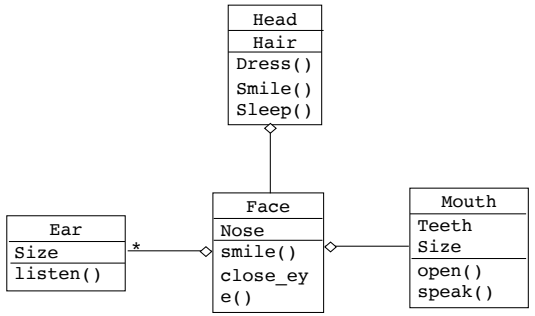
2

Possible Object Model: Eskimo



3

Alternative: Head



4

Where are we right now?

- Three ways to deal with complexity:
  - **Abstraction**
  - **Decomposition (Technique: Divide and conquer)**
  - **Hierarchy (Technique: Layering)**
- Two ways to deal with decomposition:
  - **Object-orientation and functional decomposition**
  - **Functional decomposition leads to unmaintainable code**
  - **Depending on the purpose of the system, different objects can be found**
- What is the right way?
  - **Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).**
- What activities and models are needed?
  - **This leads us to the software lifecycle we use in this class**

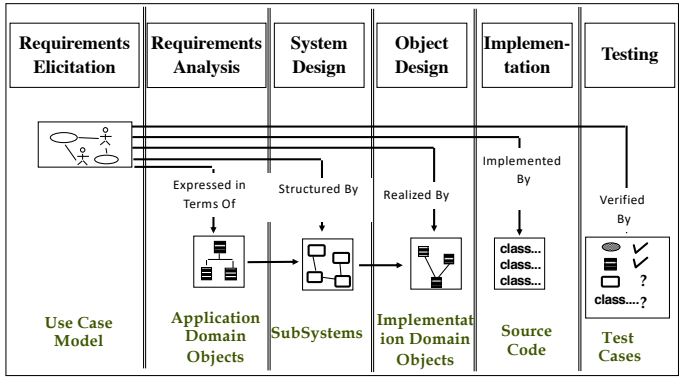
6

Software Lifecycle Definition

- Software lifecycle:
  - **Set of** activities **and their relationships to each other to support the development of a software system**
- Typical Lifecycle questions:
  - **Which activities should I select for the software project?**
  - **What are the dependencies between activities?**
  - **How should I schedule the activities?**
  - **What is the result of an activity?**

7

Software Lifecycle Activities



9

First Step in Establishing the Requirements: System Identification

- Two questions need to be answered:
  - **How can we identify the purpose of a system?**
  - **Crucial is the definition of the system boundary: What is inside, what is outside the system?**
- These two questions are answered in the requirements process
- The requirements process consists of two activities:
  - **Requirements *Elicitation*:**
    - **Definition of the system in terms understood by the *customer* ("Problem Description")**
  - **Requirements *Analysis*:**
    - **Technical specification of the system in terms understood by the *developer* ("Problem Specification")**

10

Defining the System Boundary: What do you see?

11

The System Boundary

- Which objects are inside the application domain and which ones are outside of it. Sometimes it helps you to get a clearer understanding of the overall system.
- Look at the figure in the slide. What does it show? A bunch of black and white dots? Given that I will tell you that it contains a system (that is an object model can be found) how would you start with looking for objects?
- Turn the slide around. Turn it upside down. *Look at the "problem domain" from all angles*. And suddenly you might experience what I would call the "gestalt experience". You will see the application domain.
- There is no recipe for finding it. You might find a very low level object, such as an ear or you might find a high level object such as the shape of a dog. In fact, if you look carefully you will find a dalmatian dog. Once you understand that you are looking at a dog, a lot of the black and white pixels in the total figure are not part of your system and you can easily find the boundary of the system by trying to trace the outline of the Dalmatian.
- However, *don't be lured into thinking that this is the system you have been looking for*. Always *be alert* that the real system might be something totally different. For example, if you turn the dog upside down, you might be able to see an eagle taking off from a river, with a poor dead victim in its claws!

12

Example of an Ambiguous Specification

During an experiment, a laser beam was directed from earth to a mirror on the Space Shuttle Discovery

The laser beam was supposed to be reflected back towards a mountain top 10,023 feet high

The operator entered the elevation as "10023"

The light beam never hit the mountain top
What was the problem?

The computer interpreted the number in miles...

13

Example of an Unintended Feature

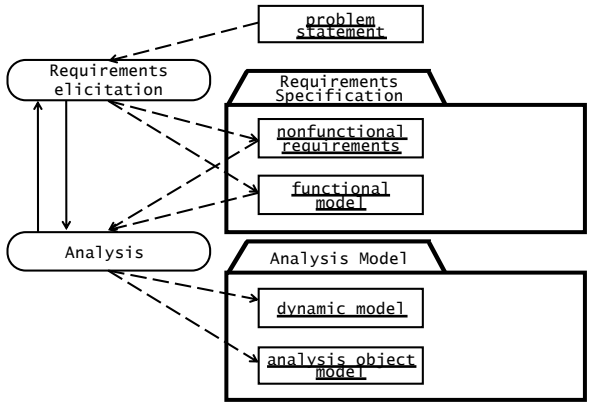
From the News: London underground train leaves station without driver!

What happened?

- A passenger door was stuck and did not close
- The driver left his train to close the passenger door
  - He left the driver door open
  - He relied on the specification that said the train does not move if at least one door is open
- When he shut the passenger door, the train left the station without him.
  - The driver door was not treated as a door in the source code!

14

Products of Requirements Process



15

System Specification vs Analysis Model

- Both models focus on the requirements from the user's view of the system.
- ***System specification*** uses natural language (derived from the *problem statement*)
- The ***analysis model*** uses formal or semi-formal notation (for example, a graphical language like UML)
- The starting point is the problem statement

16

Problem Statement

- The problem statement is developed by the client as a description of the problem addressed by the system
- Other words for problem statement:
  - **Statement of Work**
- A **good** problem statement describes
  - **The current situation**
  - **The functionality the new system should support**
  - **The environment in which the system will be deployed**
  - **Deliverables expected by the client**
  - **Delivery dates**
  - **A set of acceptance criteria**

17

Ingredients of a Problem Statement

- Current situation: The Problem to be solved
- Description of one or more *scenarios*
- Requirements
  - **Functional and Nonfunctional requirements**
  - **Constraints ("pseudo requirements")**
- Project Schedule
  - **Major milestones that involve interaction with the client including deadline for delivery of the system**
- Target environment
  - **The environment in which the delivered system has to perform a specified set of system tests**
- Client Acceptance Criteria
  - **Criteria for the system tests**

18

Current Situation: The Problem To Be Solved

- There is a problem in the current situation
  - **Examples:**
    - **The response time when playing letter-chess is far too slow.**
    - **I want to play Go, but cannot find players on my level.**
- What has changed? Why can address the problem now?
  - **There has been a change, either in the application domain or in the solution domain**
  - *Change in the application domain*
    - **A new function (business process) is introduced into the business**
    - **Example: We can play highly interactive games with remote people**
  - *Change in the solution domain*
    - **A new solution (technology enabler) has appeared**
    - **Example: The internet allows the creation of virtual communities.**

19

ARENA: The Problem

- The Internet has enabled virtual communities
  - **Groups of people sharing common of interests but who have never met each other in person. Such virtual communities can be short lived (e.g people in a chat room or playing a multi player game) or long lived (e.g., subscribers to a mailing list).**
- Many multi-player computer games now include support for virtual communities.
  - **Players can receive news about game upgrades, new game levels, announce and organize matches, and compare scores.**
- Currently each game company develops such community support in each individual game.
  - **Each company uses a different infrastructure, different concepts, and provides different levels of support.**
- This redundancy and inconsistency leads to problems:
  - **High learning curve for players joining a new community,**
  - **Game companies need to develop the support from scratch**
  - **Advertisers need to contact each individual community separately.**

20

ARENA: The Objectives

- Provide a generic infrastructure for operating an arena to
  - **Support virtual game communities.**
  - **Register new games**
  - **Register new players**
  - **Organize tournaments**
  - **Keeping track of the players scores.**
- Provide a framework for tournament organizers
  - **to customize the number and sequence of matchers and the accumulation of expert rating points.**
- Provide a framework for game developers
  - **for developing new games, or for adapting existing games into the ARENA framework.**
- Provide an infrastructure for advertisers.

21

Types of Requirements

- Functional requirements:
  - **Describe the interactions between the system and its environment independent from implementation**
  - **Examples:**
    - **An ARENA operator should be able to define a new game.**
- Nonfunctional requirements:
  - **User visible aspects of the system not directly related to functional behavior.**
  - **Examples:**
    - **The response time must be less than 1 second**
    - **The ARENA server must be available 24 hours a day**
- Constraints ("Pseudo requirements"):
  - **Imposed by the client or the environment in which the system operates**
    - **The implementation language must be Java**
    - **ARENA must be able to dynamically interface to existing games provided by other game developers.**

22

What is usually not in the requirements?

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

- It is desirable that none of these above are constrained by the client. Fight for it!

23

Requirements Validation

- Requirements validation is a critical step in the development process, usually after requirements engineering or requirements analysis. Also at delivery (client acceptance test).
- **Requirements validation criteria:**
  - **Correctness:**
    - The requirements represent the client's view.
  - **Completeness:**
    - **All** possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system
  - **Consistency:**
    - There are NO functional or nonfunctional requirements that *contradict* each other
  - **Realism:**
    - Requirements can be implemented and delivered
  - **Traceability:**
    - Each system function can be *traced* to a corresponding set of functional requirements

24

Requirements Evolution

- Problem with requirements validation: Requirements change very fast during requirements elicitation.
- Tool support for managing requirements:
  - **Store requirements in a shared repository**
  - **Provide multi-user access**
  - **Automatically create a system specification document from the repository**
  - **Allow change management**
  - **Provide traceability throughout the project lifecycle**
- RequisitPro from Rational
  - **http://www.rational.com/products/reqpro/docs/datasheet.html**

25

Prioritizing Requirements

- High priority
  - Addressed during analysis, design, and implementation
  - A high-priority feature must be demonstrated
- Medium priority
  - Addressed during analysis and design
  - Usually demonstrated in the second iteration
- Low priority
  - Addressed only during analysis
  - Illustrates how the system is going to be used in the future with not yet available technology.

26

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**

27

Requirements Elicitation

- Very challenging activity
- Requires collaboration of people with different backgrounds
  - **Users with application domain knowledge**
  - **Developer with solution domain knowledge (design knowledge, implementation knowledge)**
- Bridging the gap between user and developer:
  - *Scenarios:* **Example of the use of the system in terms of a series of interactions between the user and the system**
  - *Use cases:* **Abstraction that describes a class of scenarios**

28

Tecnica	Serve per	Vantaggi	Svantaggi
Questionari	Rispondere a domande specifiche.	Si possono raggiungere molte persone con poco sforzo.	Vanno progettati con grande accuratezza, in caso contrario le risposte potrebbero risultare poco informative. Il tasso di risposta può essere basso.
Interviste individuali	Esplorare determinati aspetti del problema e determinati punti di vista.	L'intervistatore può controllare il corso dell'intervista, orientandola verso quei temi sui quali l'intervistato è in grado di fornire i contributi più utili.	Richiedono molto tempo. Gli intervistati potrebbero evitare di esprimersi con franchezza su alcuni aspetti delicati.
Focus group	Mettere a fuoco un determinato argomento, sul quale possono esserci diversi punti di vista.	Fanno emergere le aree di consenso e di conflitto. Possono far emergere soluzioni condivise dal gruppo.	La loro conduzione richiede esperienza. Possono emergere figure dominanti che monopolizzano la discussione.
Osservazioni sul campo	Comprendere il contesto delle attività dell'utente.	Permettono di ottenere una consapevolezza sull'uso reale del prodotto che le altre tecniche non danno.	Possono essere difficili da effettuare e richiedere molto tempo e risorse.
Suggerimenti spontanei degli utenti	Individuare specifiche necessità di miglioramento di un prodotto.	Hanno bassi costi di raccolta. Possono essere molto specifici.	Hanno normalmente carattere episodico.
Analisi della concorrenza e delle best practices	Individuare le soluzioni migliori adottate nel settore di interesse	Evitare di "reinventare la ruota" e ottenere vantaggio competitivo	L'analisi di solito è costosa (tempo e risorse)

http://www.polillo.it/facciadausare/Cap-7.htm

29

Why Scenarios and Use Cases?

- Utterly comprehensible by the user
  - **Use cases model a system from the users' point of view (functional requirements)**
    - **Define every possible event flow through the system**
    - **Description of interaction between objects**
- Great tools to manage a project. Use cases can form basis for whole development process
  - **User manual**
  - **System design and object design**
  - **Implementation**
  - **Test specification**
  - **Client acceptance test**
- An excellent basis for incremental & iterative development
- Use cases have also been proposed for business process reengineering (Ivar Jacobson)

30

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.
- Scenarios can have many different uses during the software lifecycle

31

Types of Scenarios

- As-is scenario:
  - **Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.**
    - **Example: Description of Letter-Chess**
- Visionary scenario:
  - **Used to describe a future system. Usually used in greenfield engineering and reengineering projects.**
  - **Can often not be done by the user or developer alone**
    - **Example: Description of an interactive internet-based Tic Tac Toe game tournament.**
- Evaluation scenario:
  - **User tasks against which the system is to be evaluated.**
    - **Example: Four users (two novice, two experts) play in a TicTac Toe tournament in ARENA.**
- Training scenario:
  - **Step by step instructions that guide a novice user through a system**
    - **Example: How to play Tic Tac Toe in the ARENA Game Framework.**

32

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**

33

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?**
- However, don't rely on *questionnaires* alone.
- 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**

34

Example: Accident Management System

- What needs to be done to report a "Cat in a Tree" incident?
- What do you need to do if a person reports "Warehouse on Fire?"
- Who is involved in reporting an incident?
- What does the system do if no police cars are available? If the police car has an accident on the way to the "cat in a tree" incident?
- What do you need to do if the "Cat in the Tree" turns into a "Grandma has fallen from the Ladder"?
- Can the system cope with a simultaneous incident report "Warehouse on Fire?"

35

Scenario Example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.
- John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

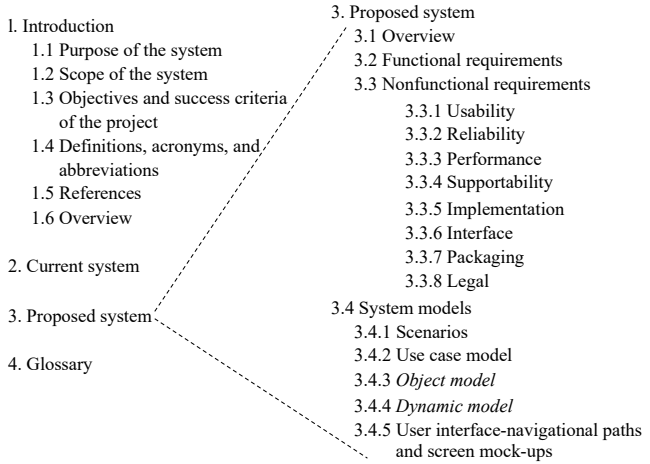
36

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - **Describes a single instance of reporting a fire incident.**
  - **Does not describe all possible situations in which a fire can be reported.**
- Participating actors
  - **Bob, Alice and John**

37

Requirements Analysis Document



38