**Chapter 1**

**Software engineering**: a collection of techniques, methodologies and tools that help with the production of a highly qualified software system developed with a given budget before a given deadline while changes occur

**Modeling:** an abstract representation of a system that enables us to answer questions about the system

**OCL:** Object Constraint Language

**UML:** Unified Modeling Language

**Configuration Management –** base set of functionality, developers track changes, management requires approval for changes.

**Bananaware** – delivering software before a standard has been set – risking it being obsolete soon after release.

**Vaporware** – software that is announced to the public but that is never actually manufactured.

**Laws of Project Management**: Projects progress quickly until they are 90% complete. Then they remain at 90% complete forever If project content is allowed to change freely, the rate of change will exceed the rate of progress. Project teams detest progress reporting because it manifests their lack of progress

**Project:** an undertaking, limited in time to achieve a set of goals that require a concerted effort. Includes a set of deliverables to a client, a schedule, technical and managerial activities required to produce and deliver the good and finally resources used to accomplish the project (people, budget).

**Project management goals**: administer the resources, maintain accountability, react to change and make sure the goals are met.

**Project organization**: defines the relationships among resources, in particular the participants of a project. It establishes a chain of command. (who decides , who reports to who)

**Role**: a set of responsibilities in a project

Ex. Role: Tester :their responsibilities are to write tests, report failures

**Task:** describes the smallest amount of work tracked by management (2-10 days)

Ex: building a house the tasks are: survey, excavate, buy material…..

**Work package**: specifies the work to be done within a task

**Completion Criteria:** a requirement of a task that defines what must be done in order for a task to be considered complete

**Work Product**: Visible outcome of a task ex. Document, piece of code etc

**Deliverables:** work products delivered to client

**Activity:** a set of tasks that is performed towards a specific purpose, can also be called phases

Ex. Phase one of house building includes the tasks: survey, excavate, buy materials, lay foundation

Ex. Each of these are activities: Planning, system design, object design, testing etc

**Chapter 4**

**Requirements Engineering:** aims at defining the system under construction. Composed of two activities: requirements elicitation and analysis

**Requirement Elicitation:** this activity includes obtaining and validating requirements and domain knowledge from the clients and users. (what are the requirements set by the client)

**Analysis: activity** Definition of the system in terms understood by the developer (Technical specification, “Analysis model”)

**Software Lifecycle:** models the development of the software

**Techniques to elicit Requirements:** (ways to obtain requirements from client)

**Questionnaires:**Asking the end user a list of pre-selected questions

**Task Analysis:**Observing end users in their operational environment

**Scenarios:** Describe the use of the system as a series of interactions between a specific end user and the system

**Use cases:** Abstractions that describe a class of scenarios.

**Functional Requirements:** Describe the interaction between the system and its environment (phrased as actions)

Ex. Create observation, edit observation, save observation

**Nonfunctional Requirements**: aspects not directly related to functional behavior (phrased as a constraints ex. All users should…. System crash should not result in)

Ex. Data collected by the application about the excursion, if made public, is saved in the cloud database for other users to download and use.

**Constraints:** Imposed by the client or the environment

Ex. The code must be written in java

**Usability:** The ease with which actors can perform a function in a system

**Robustness:** The ability of a system to maintain a function (ex. App doesn’t crash when loss of signal occurs)

**Availability:** The ratio of the expected uptime of a system to the aggregate of the expected up and down time (updates will occur once a week and during this time the system will be down)

**Requirement Validation:** is a quality assurance step, usually after requirements elicitation or analysis.

**Correctness**: The requirements represent the client’s view

**Completeness:** All possible scenarios, in which the system can be used, are described

**Consistency:** There are no requirements that contradict each other.

**Clarity:** Requirements can only be interpreted in one way

**Realism**: Requirements can be implemented and delivered

**Traceability:** Each system component and behavior can be traced to a set of functional requirements

**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.

**Requirements Analysis Document Template**

1. Introduction

2. Current system

3. Proposed system

3.1 Overview

3.2 Functional requirements

3.3 Nonfunctional requirements

3.4 Constraints (“Pseudo requirements”)

3.5 System models

3.5.1 Scenarios

3.5.2 Use case model

3.5.3 Object model

3.5.3.1 Data dictionary

3.5.3.2 Class diagrams

3.5.4 Dynamic models

3.5.5 User interface

4. Glossary

**Chapter 2:**

**Functional model:** represented in UML with use case diagrams, describes functionalities from the users point of view

**Object model :** represented in UML with class diagrams, describes the structure of the system in terms of objects , attributes, associations, and operations

**Dynamic Model:** represented in UML with interaction diagrams , state the machine diagrams and activity diagrams, describes internal behavior of the system.

**Chapter 5**

**Steps of object modeling**

1. Class identification (based on abstractions)
2. Find the attributes (data that is stored about an object)
3. Find the methods (actions that the objects can do)
4. Find the associations between classes

**Communication Diagram:**

* Shows the temporal relationship among objects
* Position of objects is identical to the position of the classes in the corresponding UML class diagram
* Good for identifying the protocol between objects
* Does not show time

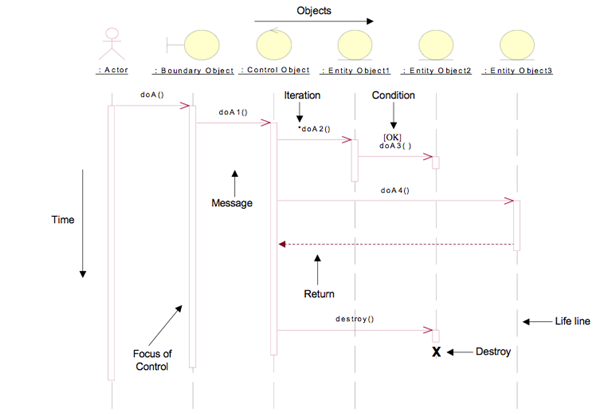
**Sequence Diagram:**

* Describes the dynamic behavior *between* several objects over time
* Sequence diagrams depict the events that occur between objects in a system.
* Good for real-time specifications.

**Boundary objects:** classes handle the communication with external actors.

Examples: windows, screens, menus.Boundary objects usually do not interact with entity objects.

**Control object:** classes are transaction classes that execute a sequence of operations.

**Entity object** classes represent real-life objects or concepts that are internal to the system

**10 Step Heuristic for Designing Sequence Diagrams**

1. Select the initiating actor and initiating event from the use case description.
2. Identify the primary display screen needed for implementing the use case. Call it the *primary boundary object.*
3. Create a *use-case controller* (*primary control object)* to handle communication between the primary boundary object and domain objects.
4. If the use case involves any included or extended use cases, create one *secondary control object* for each of them.
5. Identify the number of major screens necessary to implement the use case. Create one *secondary boundary object* for each of the major screens and create one *secondary control object* for each of them.
6. From the class diagram, list all domain classes participating in the use case by reviewing the use case description. If any class identified from the use case description does not exist in the class diagram, add it to the class diagram
7. Use those classes just identified as block labels (column names) in the sequence diagram. List the classes in the following order:
   1. The primary boundary object
   2. The primary use case controller
   3. Domain classes (in order of access)
   4. Secondary boundary and control objects in the order of access
8. Identify all problem solving operations based on the following classifications
   1. Instances creating and destruction
   2. Association forming
   3. Attribute modification
   4. Calculation
   5. Change states
   6. Display or reporting requirements
   7. Interface with external objects or systems
9. Rearrange the sequence of messages among the object classes based on any pre-existing design patterns, when possible.
10. Name each message and supply it with optional parameters. This can be done at design stage as well.

Chapter 11:

**Failure:** Any deviation of the observed behavior from the specified behavior

**Erroneous state (error):** The system is in a state such that further processing by the system can lead to a failure (ex. Wrong user input)

**Fault:** The mechanical or algorithmic cause of an error (“bug”)

**Validation:** Activity of checking for deviations between the observed behavior of a system and its specification.

**Fault avoidance**

* 1. Use methodology to reduce complexity
  2. Use configuration management to prevent inconsistency
  3. Apply verification to prevent algorithmic faults
  4. Use reviews to identify faults already in the design

**Fault detection**

* Testing: Activity to provoke failures in a planned way
* Debugging: Find and remove the cause (fault) of an observed failure
* Monitoring: Deliver information about state and behavior => Used during debugging

**Fault tolerance**

* Exception handling
* Modular redundancy.

**Test model**: consolidates all test related decisions and components into one package (sometimes also test package or test requirements)

**Generating the Test Model**

**Manually:** The developers set up the test data, run the test and examine the results themselves. Success and/or failure of the test is determined through observation by the developers

**Automatically:** *Automated generation* of test data and test cases. Running the test is also done automatically, and finally the comparison of the result with the oracel is also done automatically

**Testing Activities:**

**Test Planning:** allocates resources and schedules the testing.

**Usability testing:** tries to find faults in the user interface design of the system

**Unite Testing**: tries to find faults in participating objects and or subsystems

**Integrated testing:** the activity of finding faults by testing individual components in combination

**Systems testing:** tests all the components together, seen as a single system to identify faults with respect to the scenario of the problem statement.

**Test Doubles:** A test double is used if the collaborator in the system model is awkward to work with

* **Dummy object**: Passed around but never actually used. Dummy objects are usually used to fill parameter lists
* **Fake object**: A fake object is a working implementation, but usually contains some type of “shortcut” which makes it not suitable for production code (Example: A database stored in memory instead of a real database)
* **Stub**: Provides canned answers to calls made during the test, but is not able to respond to anything outside what it is programmed for
* **Mock object**: Mocks are able to mimic the behavior of the real object. They know how to deal with sequence of calls they are expected to receive.

**Static Analysis:**

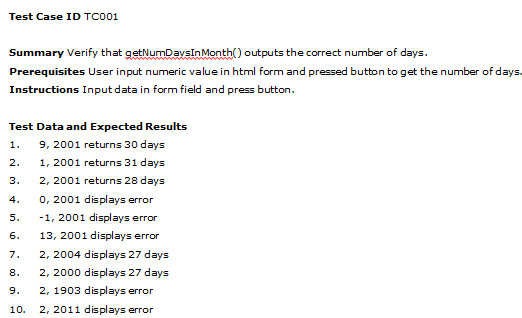
* Hand execution: Reading the source code
* Walk-Through (informal presentation to others)
* Code Inspection (formal presentation to others)
* Automated Tools checking for
  + syntactic and semantic errors
  + departure from coding standards

**Dynamic Analysis (Black –Box testing)**

* Black-box testing (Test the input/output behavior)
* White-box testing (Test the internal logic of the subsystem or class)
* Data-structure based testing (Data types determine test cases)

**Black –box Testing:**

* Focuses on the input/output behavior of the component
* Goal: Reduce number of test cases by equivalence partitioning:
  + Divide inputs into equivalence classes
  + Choose test cases for each equivalence class
  + Example: If an object is supposed to accept a negative number, testing one negative number is enough.



**White –box testing**

* Focus: Thoroughness (Coverage). Every statement in the component is executed at least once
* Four types of white-box testing: Statement Testing, Loop Testing, Path Testing, Branch Testing.
* **Statement Testing** (Algebraic Testing) : Tests each statement (Choice of operators in polynomials, etc)
* **Loop Testing**
  + Loop to be executed exactly once
  + Loop to be executed more than once
  + Cause the execution of the loop to be skipped completely
* **Path testing:** Makes sure all paths in the program are executed
* **Branch Testing** (Conditional Testing)
* Ensure that each outcome in a condition is tested at least once
* Example: **if ( i = TRUE) printf(”Yes"); else printf(”No");**
  + **We need two test cases with the following input data**
  + **1) i = TRUE**
  + **2) i = FALSE**
  + **What is the expected output for the two cases?**
  + **In both cases: Yes**
  + **This a typical beginner‘s mistake in languages, where the assignment operator also returns the value assigned ((C, Java)**
  + **So tests can be faulty as well☹**
  + **Some of these faults can be identified with static analysis.**

**Test Plan:** A Test Plan is a document that details the testing procedure

* **Introduction –** Overview, goals, constraints
* **Test Items & Environment -** List the software items and versions and test environment, tools
* **Approach –** Testing method (manual/auto, white/black box)
* **Roles –** List of responsibilities of team members
* **Schedule –** summary of schedule, specify key test milestones.
* **Test Cases –** Use unique identifiers for each test case. Each test case includes module, prerequisites, special instructions, test data, expected result.
* **References –** how to find RAD and other documents