Software Design and Architecture
Lecture 1

Dr. M. Taimoor Khan

Recommended Books

- Craig Larman: Applying UML
- Patterns, *Pearson – Prentice Hall*, 2005

- Recommended books:
  - Object Oriented Modeling and Design using UML, 2nd Edition Author(s): Simon Benett, Ray Farmer *Publisher: McGraw- Hill*
  - Introduction to object oriented analysis & design with UML and unified process, Author(s): Stephen R. Schach; *McGraw Hill*, 2004.
  - Object Oriented Analysis & Design with Applications, 2nd Edition Author(s): *Grady Booch Publisher: Addison Wesley*

Course Etiquette

- Code of conduct
  - Copying, cheating or immature behavior will not be tolerated.
- You are encouraged to attend to all lectures, (anyway, you already know attendance rules).
- You are encouraged to ask questions.
- You are also encouraged to offer answers.
- The objective of the class is to create together a learning environment.

Objectives of the Class

- Appreciate Software Engineering:
  - Build complex software systems in the context of frequent change using object oriented paradigm.
- Understand how to
  - Produce a high quality software system within time
  - While dealing with complexity and change
- Acquire technical knowledge (main emphasis)
- Acquire managerial knowledge

Course Outline

- Introduction to the course
- Software Development Process
- Concept of Object Oriented Programming
- Object-Oriented vs. Structured Analysis & Design
- Software Development Life Cycle (SDLC)
- Waterfall vs. Iterative/Incremental Software Development
- Unified Process (UP) & Phases of Unified Process
- Unified Modeling Language (UML) Activities during Inception Phase & Domain Modelling Concepts

Course Outline (contd…)

- Case studies for Use Cases and Requirements.
- Case studies covering Use Case diagrams, System Sequence Diagrams (SSD), Sequence Diagrams, Communication Diagrams, Class Diagrams.
- Implementation (Mapping Design-to-Code) Issues.
- Design Patterns i.e. General Responsibility Assignment Software Patterns (GRASP)
- System Integration and Testing.

What is 'Software Engineering'?...

- A term used occasionally in 1950s, 1960s
- Popularized in 1968 at NATO Software Engineering Conference (http://homepages.cs.ncl.ac.uk/brian.randell/NATO/)

What is Software?

- More than *computer programs.*
- The collection of programs, documentation and configuration data that ensures correct execution.
- Three major types:
  - Generic Product: Stand alone, Sold on open market.
  - Customized Product: For specific customer.
  - Embedded Product: Built into hardware.

The Nature of Software

- Intangible:
  - Opposite of physical artifacts, e.g., Computer vs *Windows XP*
  - Hard to understand the development process.
- Easy to Reproduce:
  - Costly design and construction, cheap manufacturing.
- Malleable:
  - Easy to change, even without full understanding.
  - Untrained people can “hack” something together.

Software Development Problems

- "Software is not constrained by *materials*, or governed by *physical laws*, or by *manufacturing process*" ---- (Sommerville, *Software Engineering*)
- Allows almost unbounded complexity:
  - Exponential growth of complexity w.r.t. to the size of a program: twice the size, four times the complexity;
  - **Example:** Windows XP has *40millions* lines of source code (estimation).

Software Development Problems

- Difficulty in understanding and managing the complexity causes:
  - Late completion:
    - "vaporware" that are announced but never produced
  - Overrunning Cost:
    - *Denver Airport Automated Baggage System, 2 billions US dollar* over budget
  - Unreliable
  - Maintenance
  - Etc…

Introduction to OOAD

- The proverb “owning a hammer doesn’t make one an architect” is especially true with respect to object-oriented technology
- Knowing an object language is necessary but insufficient to create object systems
- Its about analyzing system requirements in terms of objects and allocating responsibilities to class objects
- How would these objects collaborate with each other
- What classes should do what

OOA

- It emphasizes on finding requirements and problems rather than solutions
- Object oriented analysis is a process of analyzing requirements in an object oriented paradigm
- It approves some of the requirements while discards others
- The requirements are represented in the form of Use cases and personas to make more sense
- Based on this OOA the OOD is built

OOD

- It emphasizes on the conceptual solution in software or hardware that fulfills the requirements
- It does not give any implementation details
- Design ideas normally exclude low level details that are obvious to the intended customers
- Ultimately designs can be implemented in code to give its true and complete realization

OOA vs OOD

- In OOA there is an emphasis on finding and describing objects or concepts in the problem domain
- For example in flight information system some of the objects are flights, planes and pilot
- During OOD there is emphasis on defining software objects and how they collaborate with each other to fulfill requirements
- For example a flight object may have attributes like flight duration, departure time, arrival time, source, destination etc.

Use Case

- Use case consists of a paragraph or so text that presents a scenario of a single system requirement
- It is written in simple English that would help realize the importance of the requirement and its contribution to the over all system
- There can be many use cases for a system representing its various requirements
- It is not an OO activity but it helps in the development of OOD

Use Case

- Example
  - Use Case “Login to System”
  - The User accessing the system is asked for a username and password. On provision of the correct user name and password the user is allowed to enter the system.

Persona

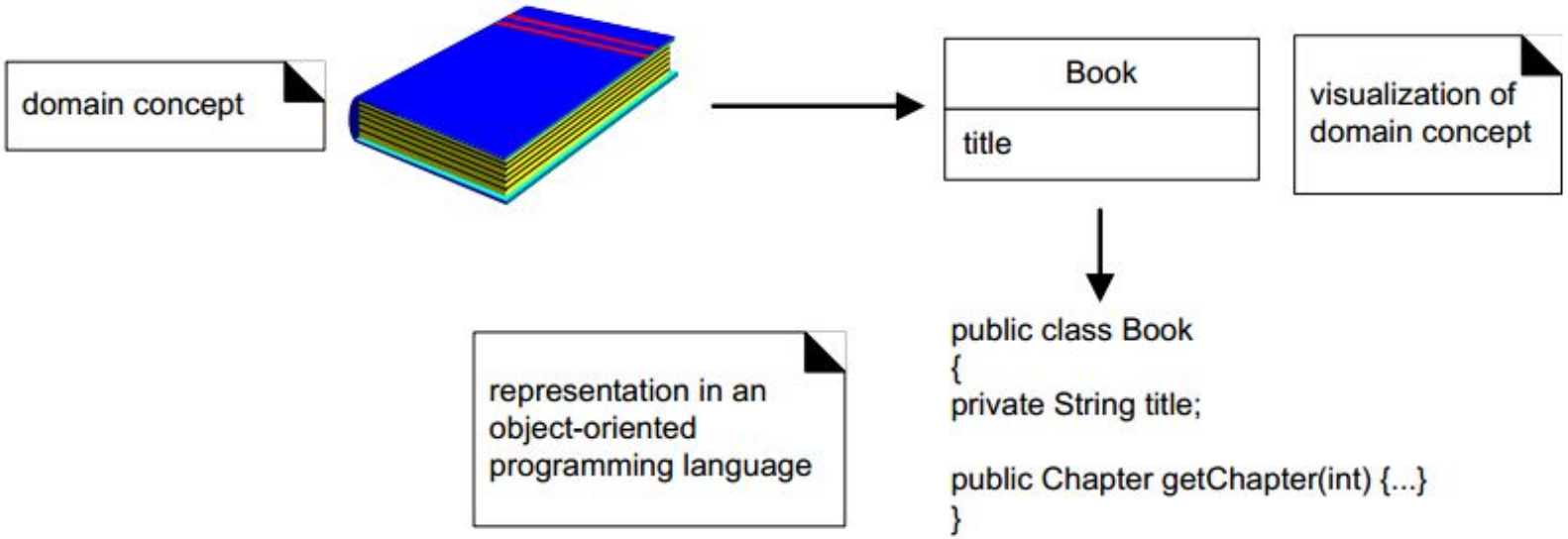
- Persona is an imaginary story in which the importance of the required system is highlighted.
- It is in the form of a short story
- It assumes a person with certain personal details whom the writer know
- The writer explains the problems of the person in the persona with the current system and how it is lagging him behind
- It also explains what the user actually wants things to be
- In this way the requirements of the system are elaborated

UML

- UML is a standard diagramming notation
- UML is not OOAD or a method it is just a diagramming notation
- It is useless to learn UML but not really know how to create an excellent OO design or evaluate and improve an existing one
- To know how to design is a separate skill

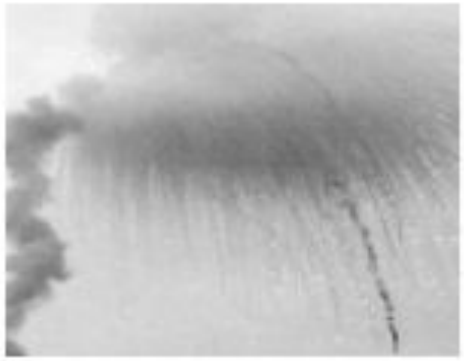
UML perspectives

- Conceptual perspective
  - The diagrams are interpreted as describing things in a situation of the real world or domain of interest
- Specification perspective
  - The diagrams (using same notations) describe software abstraction or components with specifications and interfaces but no commitment to a particular implementation
- Implementation perspective
  - Software implementation in a particular technology


domain concept
Book
title
visualization of domain concept
representation in an object-oriented programming language
public class Book
{
private String title;
public Chapter getChapter(int) {...}
}

Famous Software Disaster

- Software errors cost the U.S. economy $60 billion annually in rework, lost productivity and actual damages
- Ariane 5 expandable launch system (Spacecraft launching system was improved from Ariane 4)



- US $5 hundred million worth of payload were destroyed
- Reason:
  - Main Navigation Computer crashes after 37 seconds.
  - Caused by a conversion overflow: converting floating point number to integer number.
  - Reuse of specification of Ariane 4 without taking into consideration the new capability.

- Canada's Therac-25 radiation therapy machine malfunctioned and delivered lethal radiation doses to patients.
- Reason:
  - Because of a subtle bug called a race condition, a technician could accidentally configure Therac-25 so the electron beam would fire in high-power mode without the proper patient shielding.

EDS Child Support System

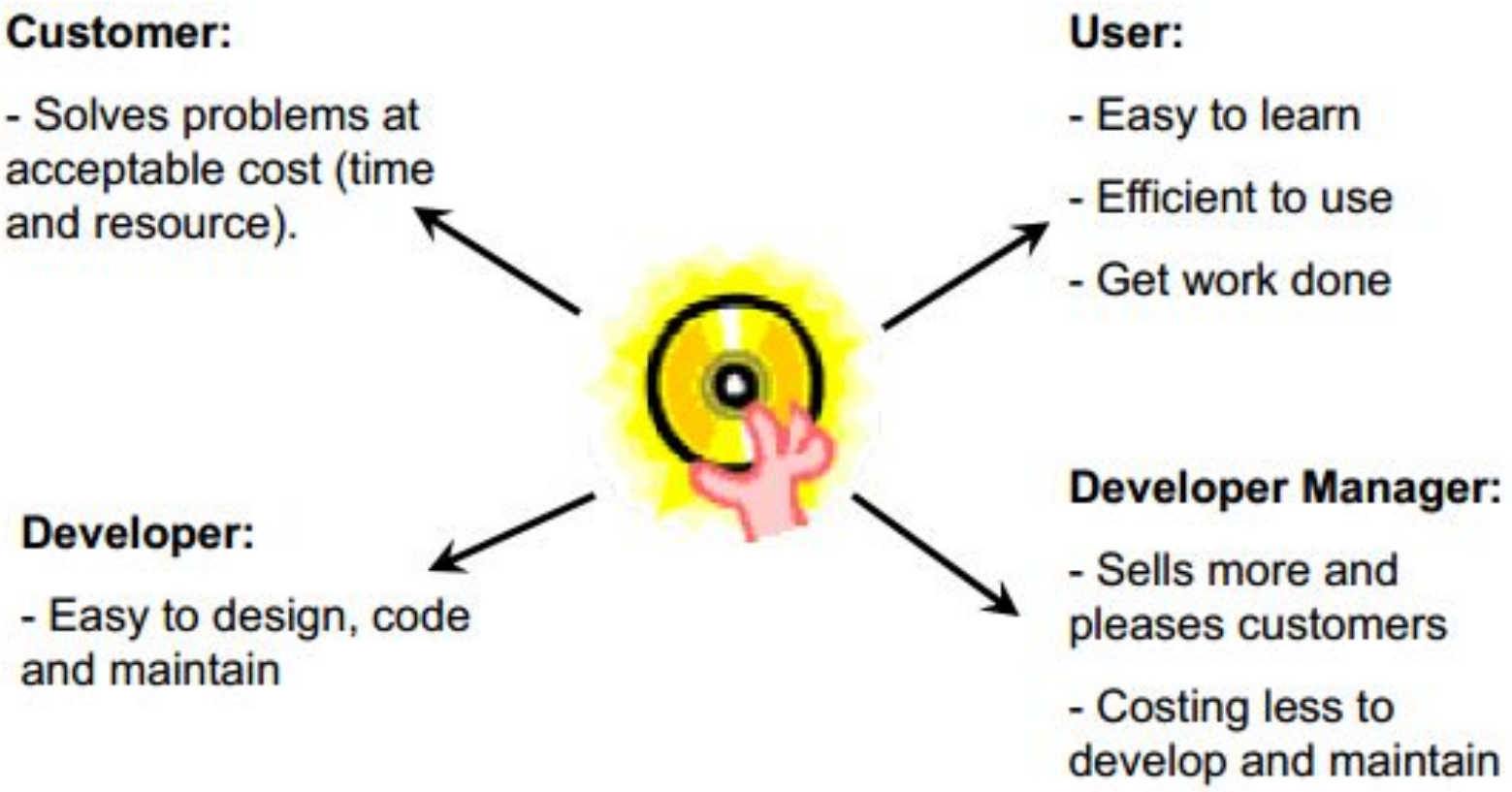
- In 2004, EDS introduced a highly complex IT system to the U.K.'s Child Support Agency
- Reason:
  - At the exact same time, the Department for Work and Pensions (DWP) decided to restructure the entire agency. The two pieces of software were completely incompatible, and irreversible errors were introduced as a result. The system somehow managed to overpay 1.9 million people, underpay another 700,000, had US$7 billion in uncollected child support payments, a backlog of 239,000 cases, 36,000 new cases "stuck" in the system, and has cost the UK taxpayers over US$1 billion to date.

Limitations of Non-Engineered Software

- One of the problems with complex system design is that you **cannot foresee the requirements** at the beginning of the project.
- In many cases, where you think you can start with a set of requirements that specifies the complete properties of your system, you end up with bugs, as well as erroneous and incomplete software.

.

Quality of Good Software



Qualities of Good Software

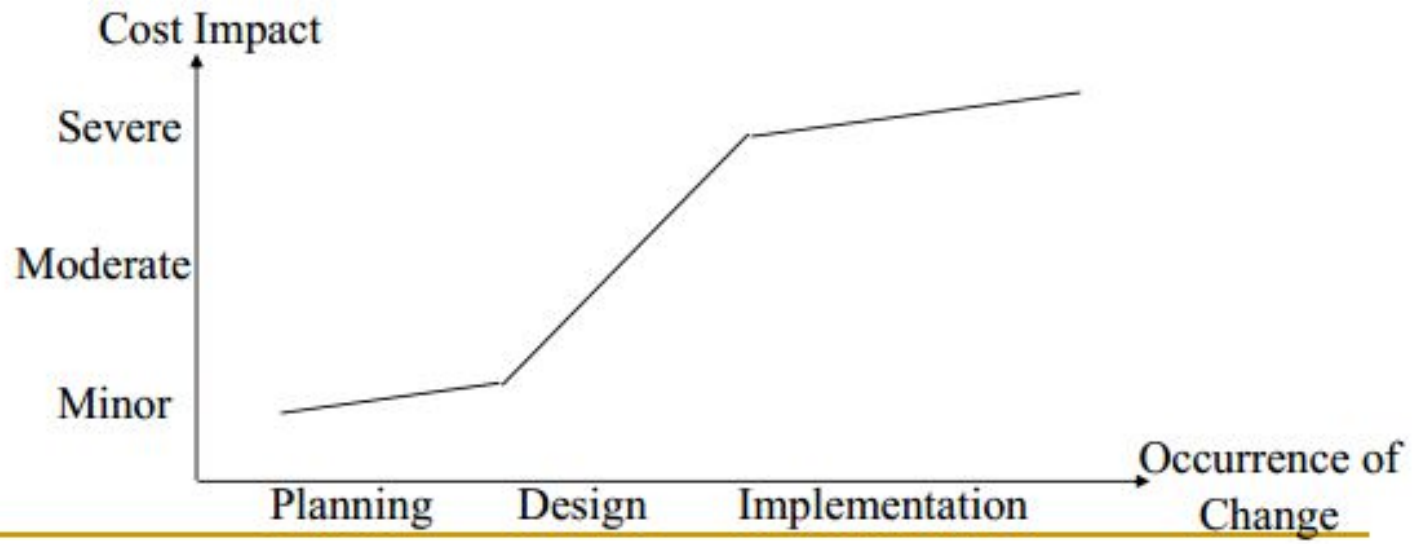
- **Usability**
  - Easy to learn and use
- **Efficiency**
- Does not waste resources such as CPU time and memory.
- **Dependability**
  - Reliable, secure and safe.
- **Maintainability**
  - Easily evolved (modified) to meet changing requirement.
- **Reusability**
  - Parts can be reused, with minor or no modification.

- IEEE Standard 610.12:
  - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software.
  - "*Designing, building and maintaining large software systems*". - I. Sommerville
  - "*Multi-person construction of multi-version software*". - D. L. Parnas

Software Engineering Myths

- A general statement of objectives is sufficient to begin writing programs - we can fill in the details later.
  - Poor Up-front definition is the major cause of failed software efforts.
- If we get behind schedule we can add more , we can add more programmers and catch up.
  - Brooks Law: “Adding people to a late project makes it later.”

- Project requirements continually change, but change can be easily accommodated because software is flexible.



Software Process Models

- The set of *activities* and *associated results* that produce a software product.
- Four fundamental process activities
  - Software Specification
  - Software Development
  - Software Validation
  - Software Evolution
- Organized differently by different Software process models having different levels of detail

1. Software Specifications

- Customers and Software Engineers define the software to be produced and the constraints on its operations. Typical Stages are,
- **Feasibility Study:**
  - Is it possible with the current technologies + within budget?
- **Domain Analysis:**
  - What is the background for the software?
- **Requirements Gathering and Analysis:**
  - What is it that the user wants?
- **Requirements Specification:**
  - Formal documentation on *User* and *System* requirements.
- **Requirements Validation:**
  - Check for realism consistency and completeness.

2. Software Development

- Consists of Design and Programming
  - System Analyst design the software and decide how the requirement can be implemented.
  - Programmers write code to translate the high level design into a real code in a chosen programming language

3. Software Validation

- Software Engineer (or dedicated tester) and Customer:
  - Check the software to ensure it meets the customers' requirements.
- Typical Stages:
  - **Component Testing:** Independent testing of individual components in subsystem.
  - **System Testing:** Testing of integrated components.
  - **Acceptance Testing:** Tested with customer supplied data. Final test before operation.

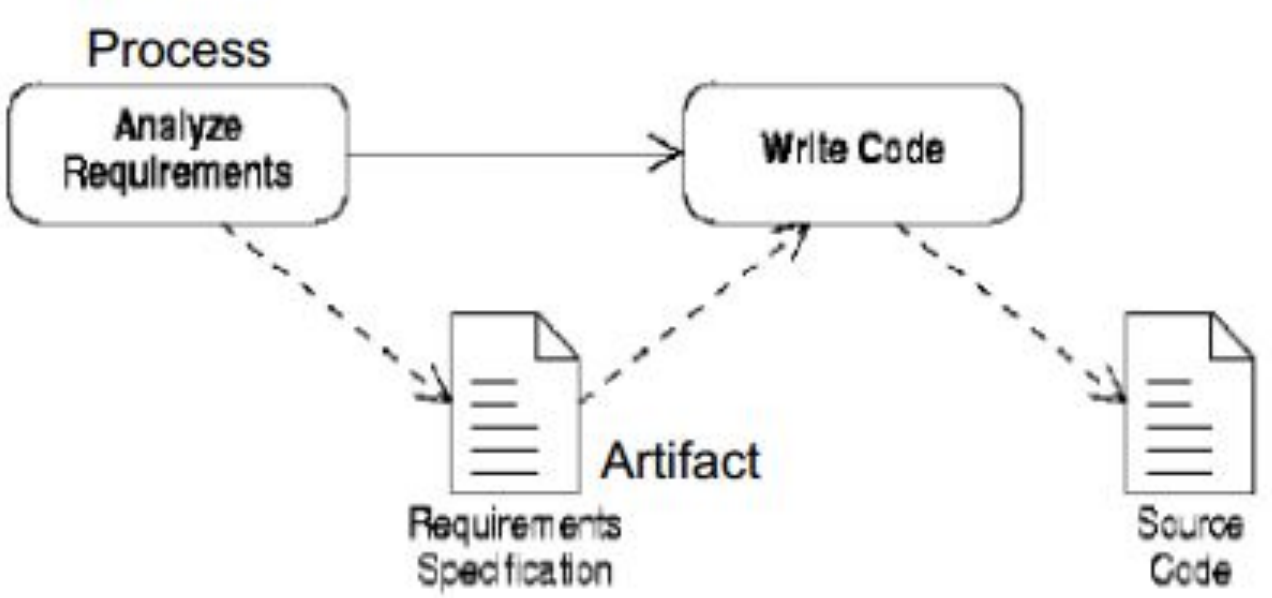
Interactive activity that feedback to previous stages: E.g., an error in component testing triggers re-coding.

4. Software Evolution

- Customers and Software Engineers:
- Define changing requirements.
  - Modify the software system to adapt.
  - Typical Work:
    Update the system for minor new requirements, e.g., changing the telephone number from 7 digits to 8 digits, changing the date representation (the *Millennium Bug*).

Simple Software Process

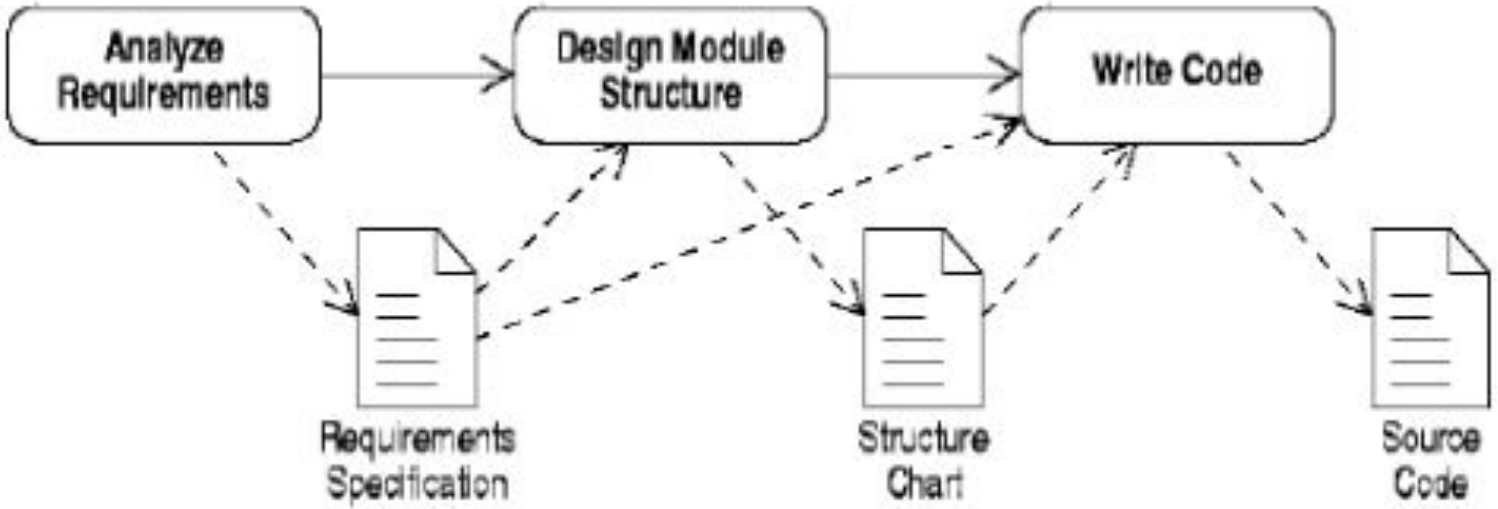
- In the simplest cases, code is written directly from some statements of requirements.



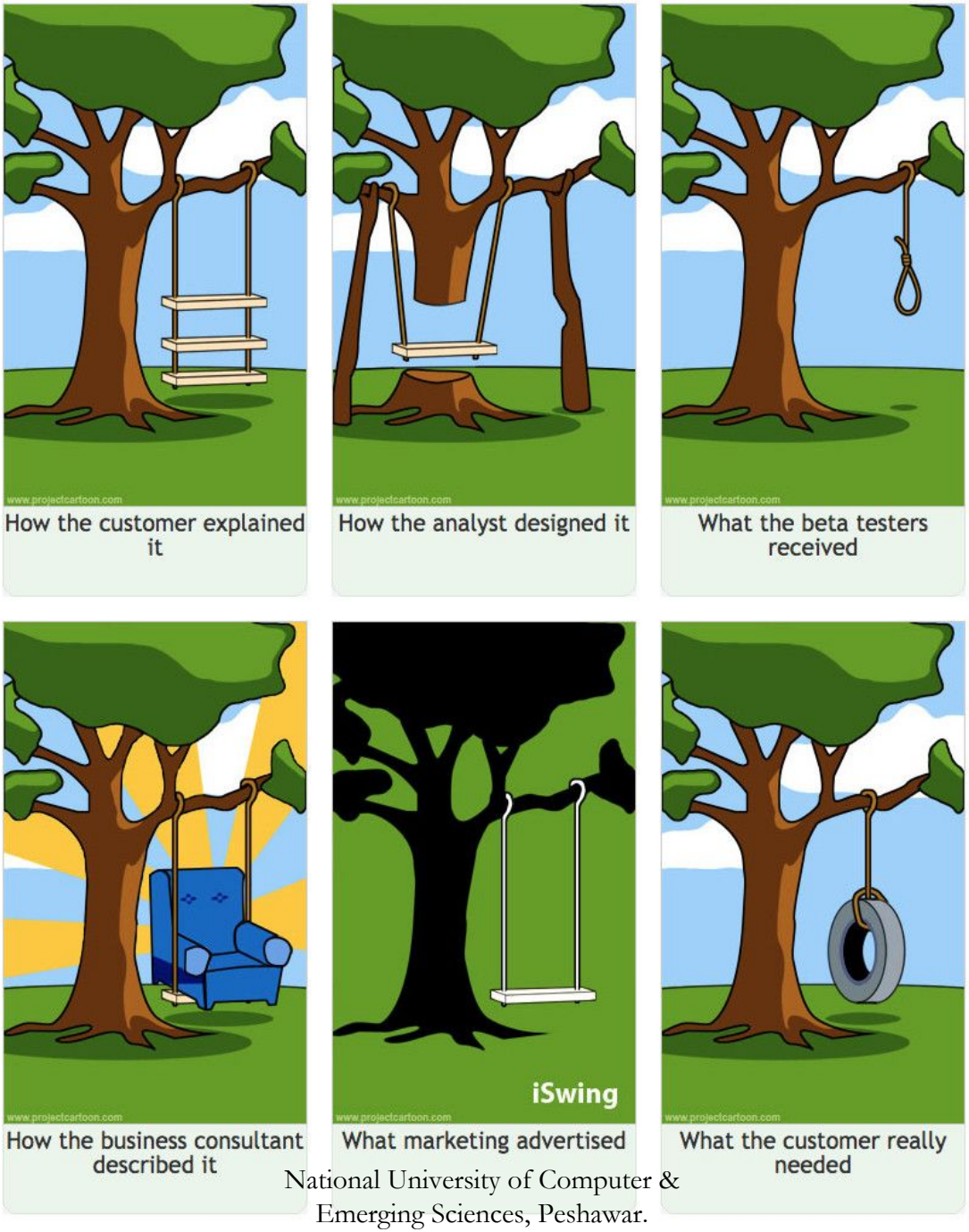
- Two processes:
  - Analyze requirements'
  - 'Write code'
- Two artifacts:
  - 'Requirements specification'
  - 'Source code'
- 'Requirements specification' can be written as:
  - an informal outline or
  - a highly detailed description.

A more Complex Software Process

- It is better to design before you code.
- On larger projects, intermediate pieces of documentation are produced.



- One new process:
  - ‘Design module structure’ - splitting the program into modules and subroutine
- One new artifact:
  - ‘Structure chart’ – is based on the information contained in the ‘requirements specification’
  - Both the ‘requirements specification’ and the ‘structure chart’ are used when writing the final code.


How the customer explained it
How the analyst designed it
What the beta testers received
How the business consultant described it
iSwing
What marketing advertised
What the customer really needed