Software Design and Analysis

LECTURE 1

Prerequisite

- **Course:**

  CS309- Object Oriented Analysis and Design

  CS324- Software Analysis and Design

- **Pre- Requisite**

  CS-218 Data Structures

Recommended Books

- Craig Larman: Applying UML and patterns
- Brett D. Mclaughlin, Gray Pollice & David West: Head First Object-Oriented Analysis and design

Evaluation Criteria

Assessment	Weightage
Sessional 1	15%
Sessional 2	15%
Quiz	10%
Assignment	10%
Project	10%
Final Exam	40%

Objective of OOAD

- This Object-Oriented Analysis & Design course teaches students how to use object-oriented techniques to build software
- The course will start with requirements gathering, use cases, UML activity diagram, sequence diagram etc. & end with implementation
- In the process, you'll learn how to analyze and design classes, their relationships to each other in order to build a model of the problem domain.

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.

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 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**
- classes should do what

Object Oriented Analysis (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

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

Object Oriented Design (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.

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

Ways to Apply UML

- Three ways to apply UML

1. UML as Sketch
2. UML as blueprint
3. UML as programming language

UML as sketch- Informal and incomplete diagrams (often hand sketched on whiteboards)created to explore difficult parts of the problem or solution space, exploiting the power of visual languages.

Ways to Apply UML

UML as blueprint - relatively detailed design diagrams used either for

1. reverse engineering to visualize and better understanding existing code in UML diagrams, or for
2. code generation(forward engineering).

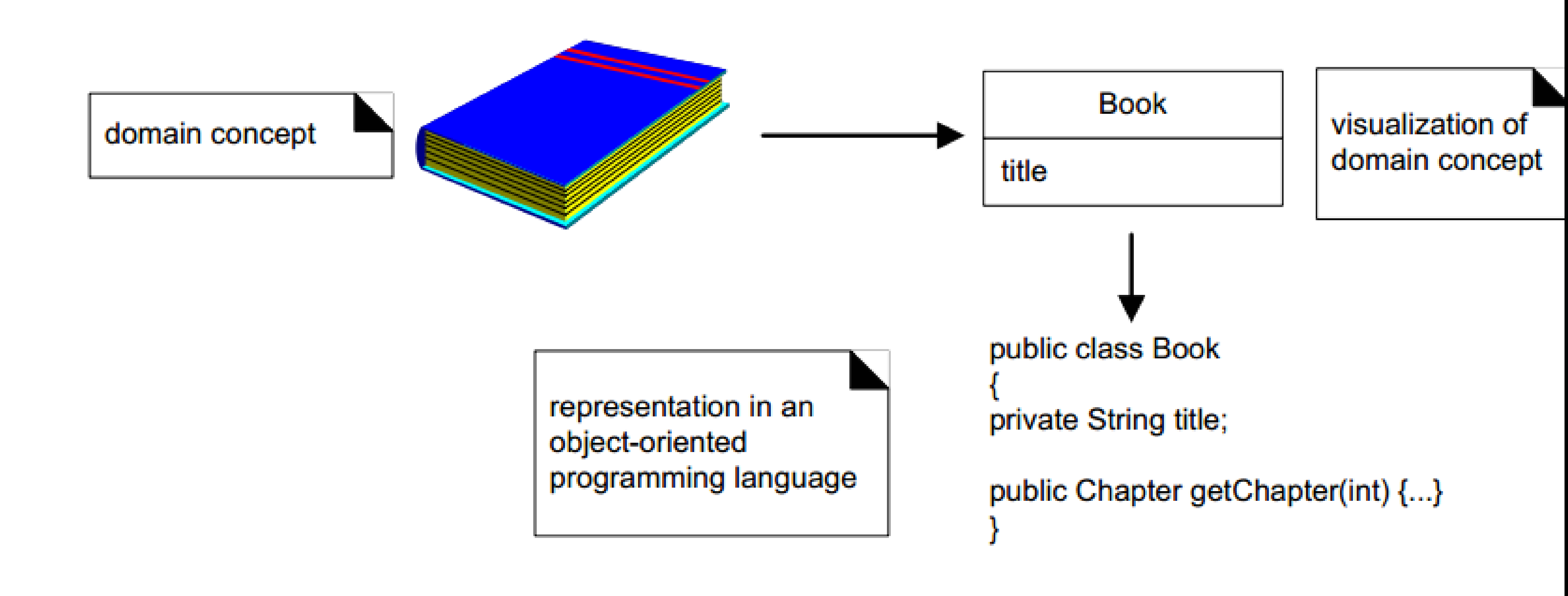
If reverse engineering, a UML tool reads the source or binaries and generates (typically) UML package, class, and sequence diagrams. These "blueprints" can help the reader understand the big picture elements, structure, and collaborations.

Ways to Apply UML

UML as programming language - Complete executable specification of a software systemin UML. Executable code will be automatically generated but is not normally seen or modified by developers; one works only in the UML "programming language."

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

- US $5 hundred million worth of payload were destroyed
- Reason:
  - Main Navigation Computer crashes after 3 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.

Famous Software Disaster

- 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.

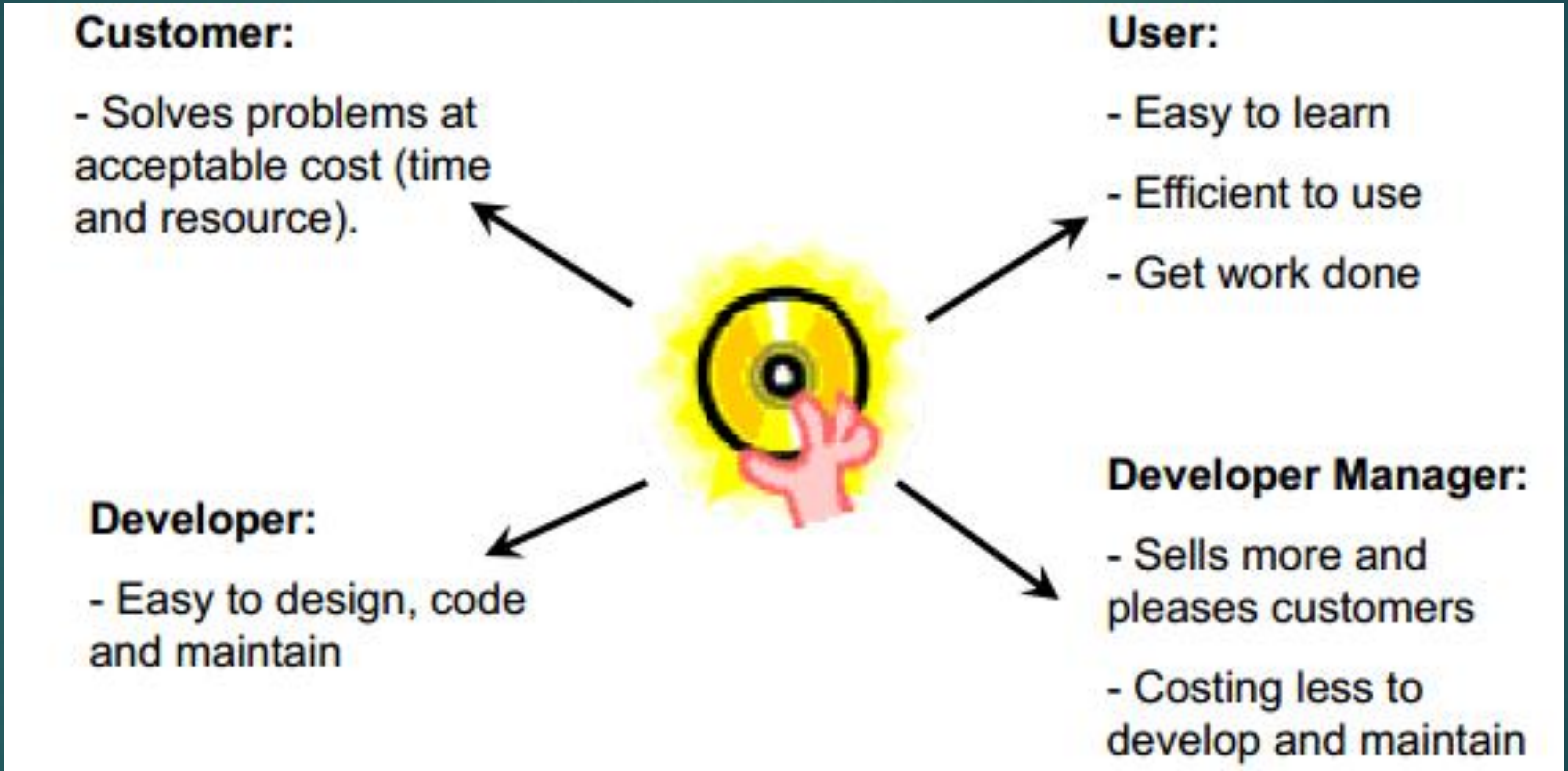
EDS Child Support System

- 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



Customer:

- Solves problems at acceptable cost (time and resource).

User:

- Easy to learn

- Efficient to use

- Get work done

Developer:

- Easy to design, code and maintain

Developer Manager:

- Sells more and pleases customers

- Costing less to develop and maintain

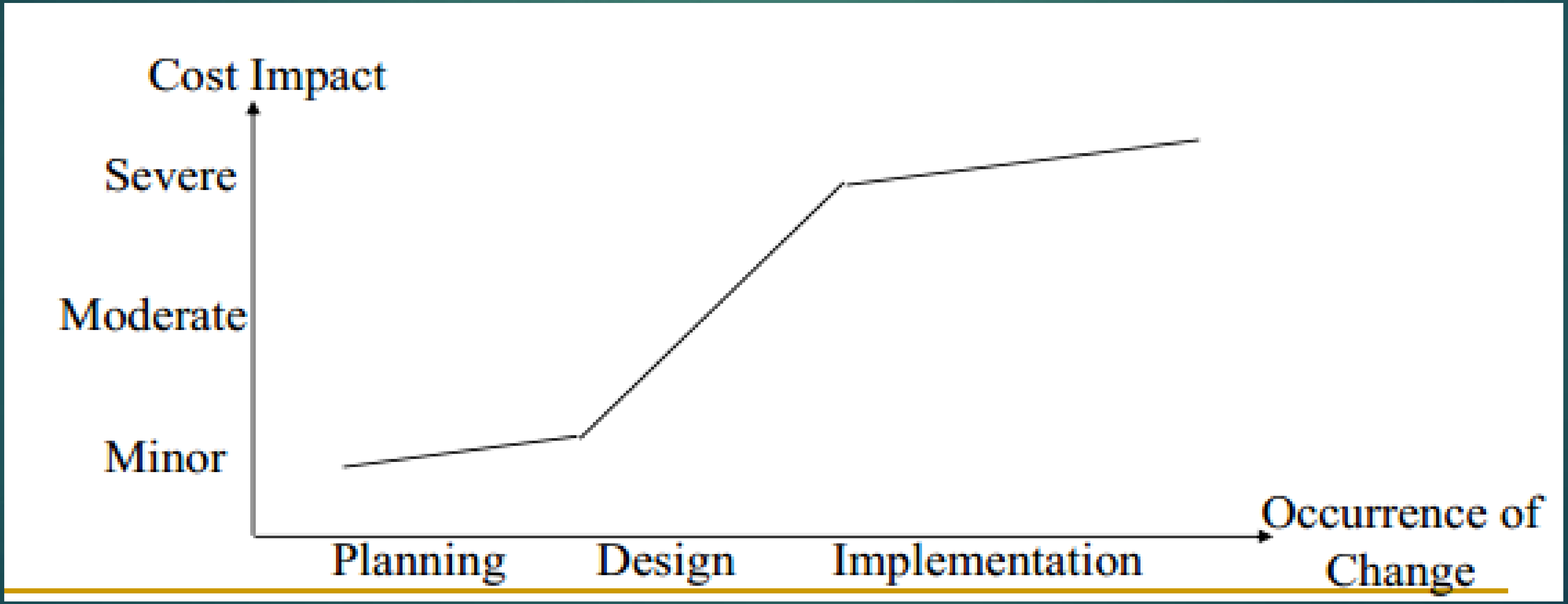
Qualities of Good Software

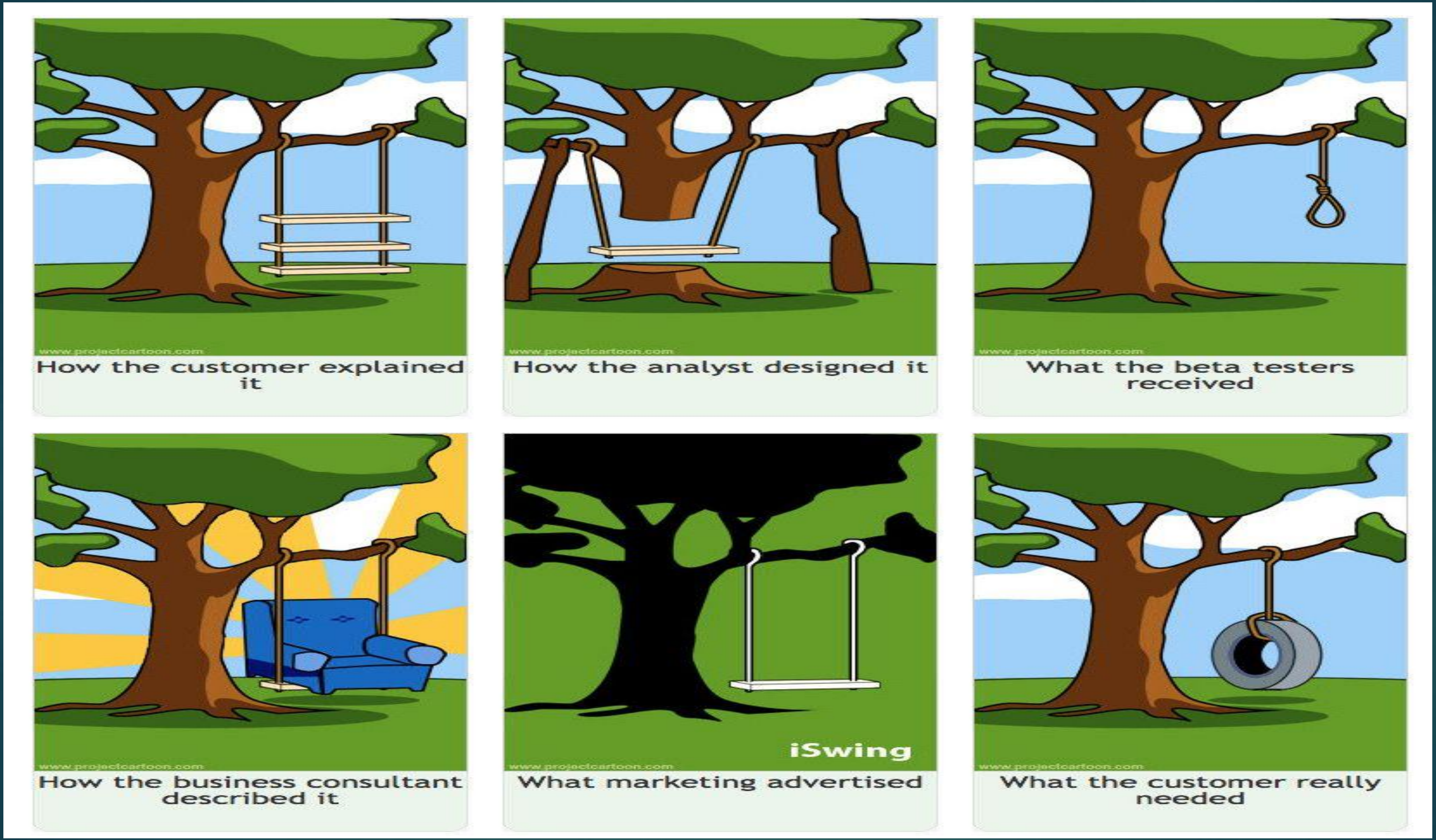
- **Usability**
  - Easy to learn and use
- **Efficiency**
  - Does not waste resources such as CPU time and memory.
- **Dependability**
  - Reliable, secure and safe.
- **Maintainability**
  - Easily evolved (modified) to meet changing requirement.
- **Reusability**
  - Parts can be reused, with minor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

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.





www.projectcartoon.com

How the customer explained it

www.projectcartoon.com

How the analyst designed it

www.projectcartoon.com

What the beta testers received

www.projectcartoon.com

How the business consultant described it

www.projectcartoon.com

What marketing advertised

www.projectcartoon.com

What the customer really needed