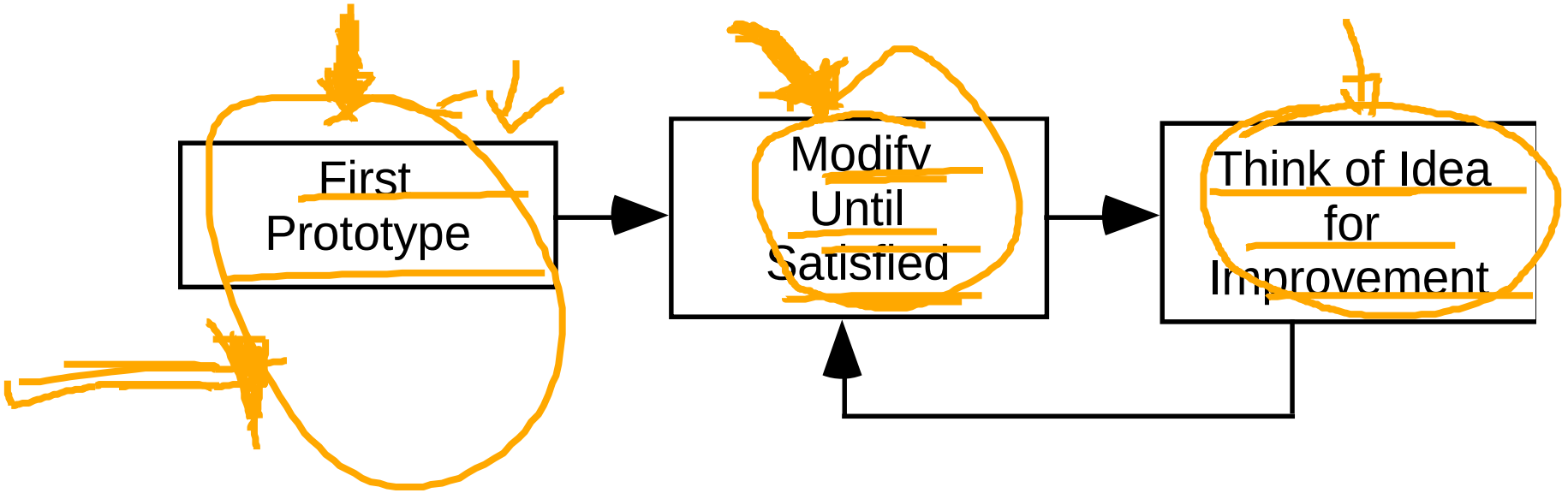
The lifecycle of Software

(Week 2)

Exploratory programming

- The early programmers used an exploratory (also called build and fix) style.
  - In the build and fix (exploratory) style, normally a `dirty' program is quickly developed.
  - The different imperfections that are subsequently noticed are fixed.

The opportunistic approach

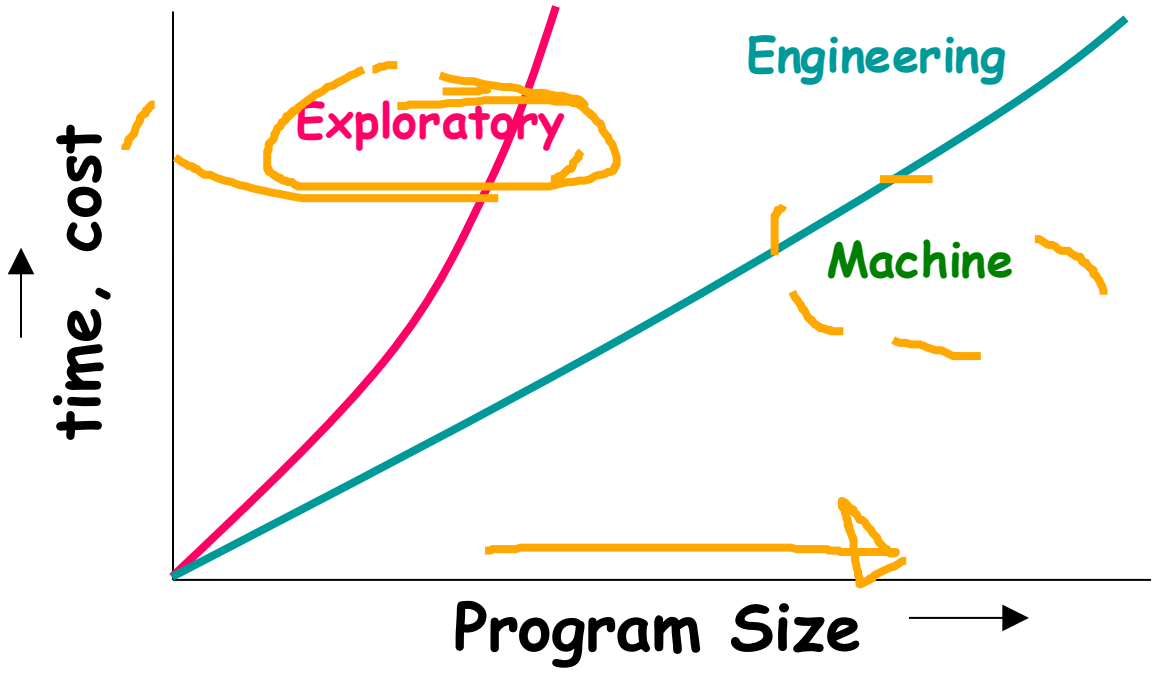


- OK for small, informal projects
- Inappropriate for professional environments/complex software where on-time delivery and high quality are expected

PROGRAMS	SOFTWARE PRODUCTS
Usually small in size	Large
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user manual prepared
Ad hoc development/Exploratory	Systematic development

What is Wrong with the Exploratory Style?

- Can successfully be used for very small programs only.



Why does the effort required to develop a product grow exponentially with product size?

What is Wrong with the Exploratory Style?

- Besides the exponential growth of effort, cost, and time with problem size:
  - Exploratory style usually results in unmaintainable code.
  - It becomes very difficult to use the exploratory style in a team development environment.
- Why does the effort required to develop a product grow exponentially with product size?

What do these have in common?

- The Boeing 777 flies with over 4,000,000 lines of code onboard.
- A typical top-level game has between 2 and 100 M SLOC (source lines of code)

➢ These are **huge** software systems
➢ Software engineering needs to think about the design at different levels from a line of code up to the entire system.

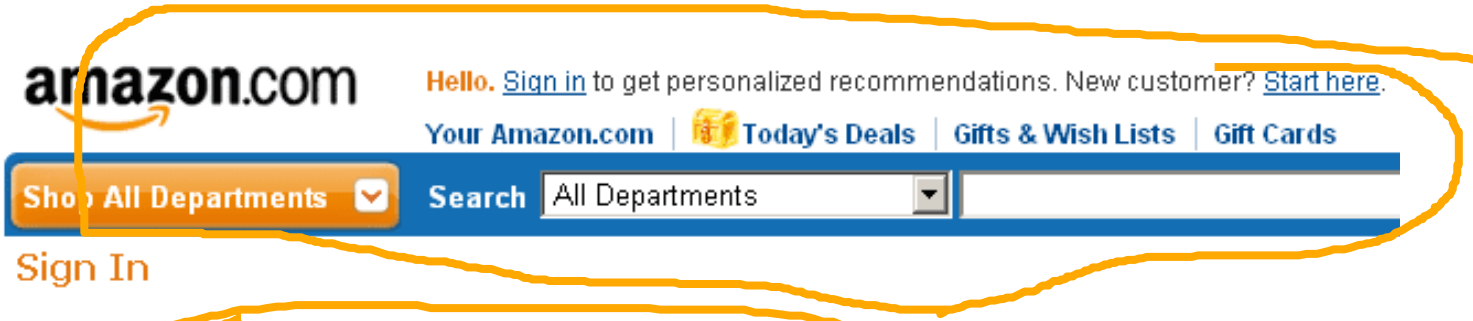
Software engineering design

1 SLOC/min/SE * 60 min/hr * 40 hrs/wk = 2,400 SLOC/wk/SE

2,400 SLOC/wk/SE * 50 wk/year = 120,000 SLOC/year/SE

6 * 10^6 SLOC / 1.2 * 10^5 SLOC/yr/SE = ~ 51 SEs for the year

Teamwork



One large ecommerce application. Complex!!!

ne small software upgrade.

asy.

ne 90 minute outage.

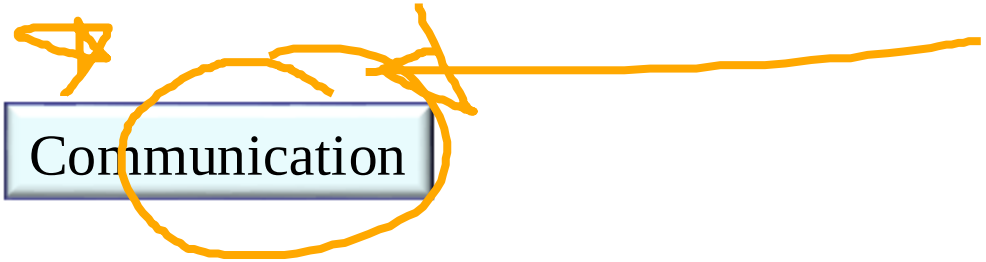
Priceless?

Teams need better understanding of the process for making changes to a financial-critical system, and how to bring an upgrade on-line without taking down the system.

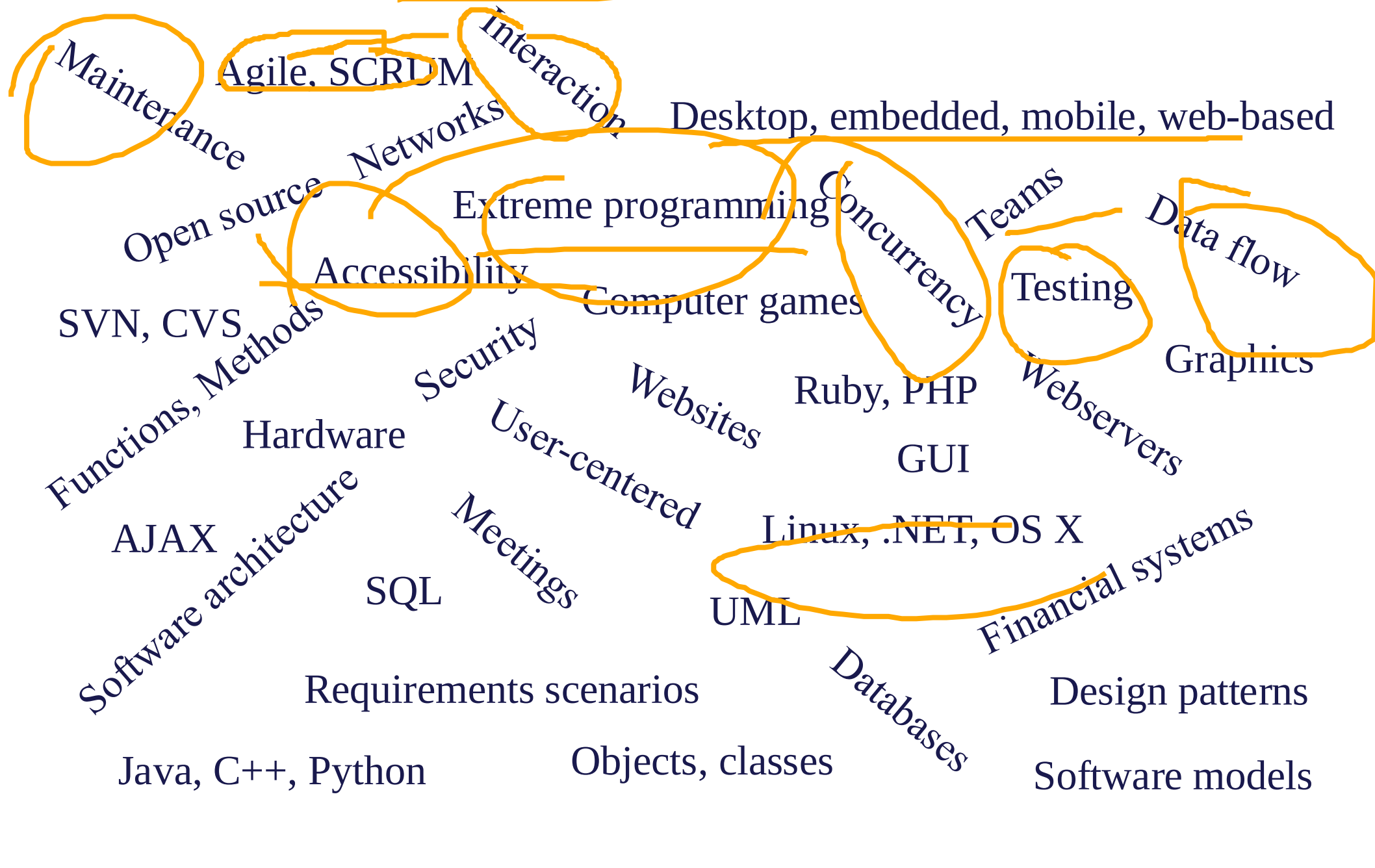
Software development process

Software engineer must work with many people…

- Customers asking for the system/parts of the system
- End-users who will use the system
- Domain experts: banking, avionics, security, medical, scientists, …
- Engineers from other engineering disciplines
- Most closely with the other software engineers on the project



Yes, software engineers get their hands dirty writing programs using the latest technologies and techniques.



Fundamental principles of SOFTWARE ENGINEERING

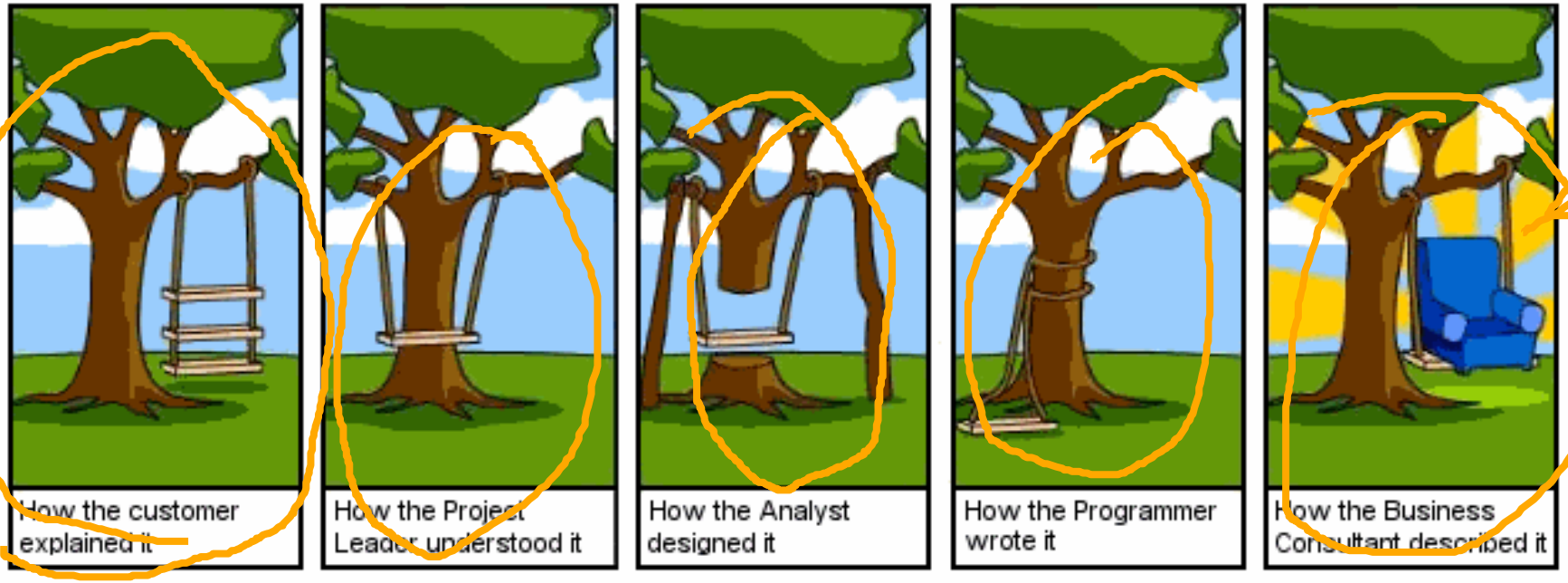
Abstraction

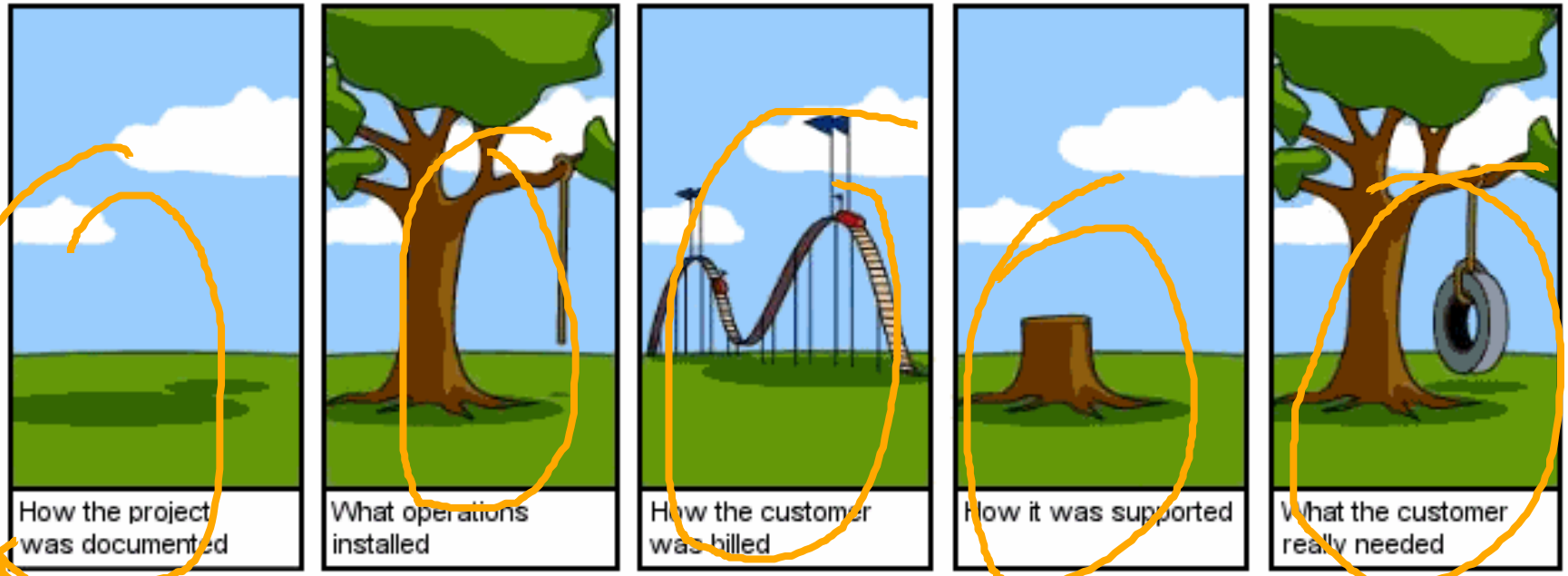
- Simplify a problem by omitting unnecessary details
- Focus attention on only one aspect of the problem and ignore irrelevant details.

Decomposition

- Decompose a problem into many small independent parts
- The small parts are then taken up one by one and solved separately
- The idea is that each small part would be easy to grasp and can be easily solved
- The full problem is solved when all the parts are solved

Software Development Life Cycle (SDLC)


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it


How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Why Life cycle model?

- A software project will never succeed if activities are not coordinated:
  - one engineer starts writing code,
  - another concentrates on writing the test document first,
  - yet another engineer first defines the file structure
  - another defines the I/O for his portion first.
- Adherence can lead to accurate status reports
- Otherwise, it becomes very difficult to track the progress of the project
  - the project manager would have to depend on the guesses of the team members.

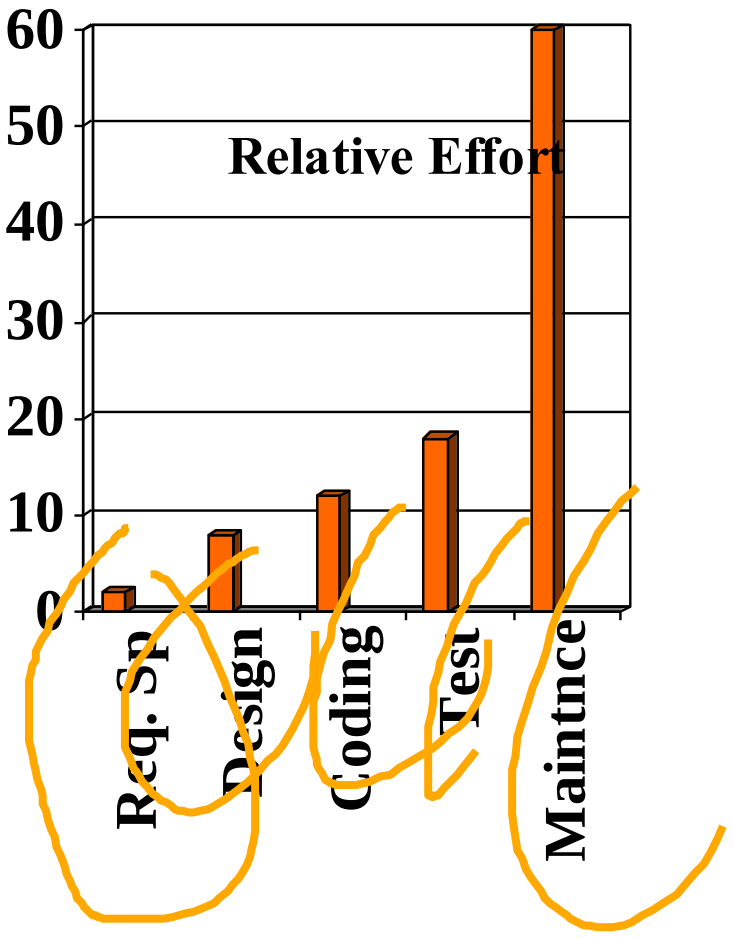
Software Development Life Cycle (SDLC)

- Typical software life cycle or software process consists of following phases:
  - Feasibility study (involves business case)
  - Requirements analysis and specification
  - Design
  - Coding
  - Testing
  - Maintenance

Relative effort for Phases

- Phases between feasibility study and testing
  - known as development phases.

- Among all life cycle phases
  - maintenance phase consumes maximum effort.



Feasibility Study

- Main aim of feasibility study: determine whether developing the product
  - financially worthwhile
  - technically feasible.

- First roughly understand what the customer wants:
  - Inputs
  - Processing
  - Outputs
  - various constraints on the behaviour of the system

Activities during Feasibility Study

- Work out an overall understanding of the problem
- Formulate different solution strategies
- Examine alternate solution strategies in terms of:
  - resources required
  - cost of development
  - development time
- Perform a cost/benefit analysis:
  - you may determine that none of the solutions is feasible due to high cost, resource constraints, technical reasons.

Requirements Analysis and Specification

- Aim of this phase:
  - understand the exact requirements of the customer,
  - document them properly.

- Consists of two distinct activities:
  - requirements gathering and analysis
  - requirements specification.

Goals of Requirements Analysis

- Collect all related data from the customer:
  - analyze the collected data to clearly understand what the customer wants,
  - ensure correctness, consistency and unambiguity.

- Does Feasibility Study as

Requirements Gathering

- Gathering relevant data:
  - usually collected from the end-users through interviews and discussions.
  - For example, for a business accounting software:
    - interview all the accountants of the organization to find out their requirements.

Requirements Analysis (CONT.)

- The data you initially collect from the users:
  - would usually contain several contradictions and ambiguities:
  - each user typically has only a partial and incomplete view of the system.
- Ambiguities and contradictions:
  - must be identified
  - resolved by discussions with the customers.
- Next, requirements are organized:
  - into a **Software Requirements Specification (SRS)** document

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.

Design

- High-level design:
  - decompose the system into *modules*,
  - represent invocation relationships among the modules.
- Detailed design:
  - different modules designed in greater detail:
    - data structures and algorithms for each module are designed.

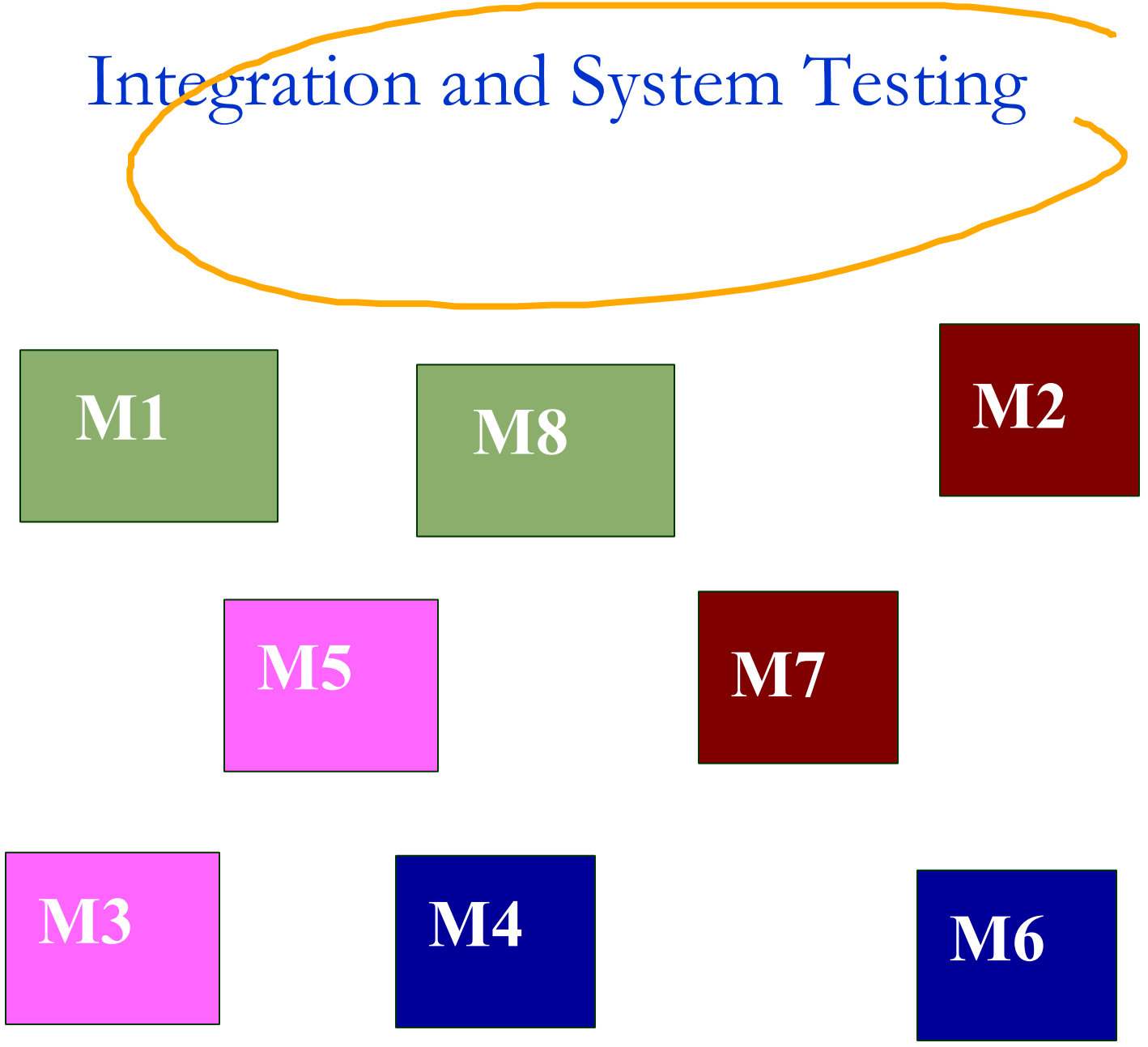
Implementation

- During the implementation phase:
  - each module of the design is coded,
  - each module is unit tested
    - tested independently as a stand alone unit, and debugged
- The purpose of unit testing:
  - test if individual modules work correctly.
- The end product of implementation phase:
  - a set of program modules that have been tested individually

Integration and System Testing

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out through a number of steps.
- During each integration step,
  - the partially integrated system is tested.

Integration and System Testing



Maintenenace

- Maintenance of any software product:
  - requires much more effort than the effort to develop the product itself.
  - development effort to maintenance effort is typically 40:60.

Maintenance (CONT.)

- Preventive maintenance
  - Making appropriate changes to prevent the occurrence of errors
- Corrective maintenance
  - Correct errors which were not discovered during the product development phases
- Perfective maintenance
  - Improve implementation of the system
  - enhance functionalities of the system
- Adaptive maintenance
  - Port software to a new environment

Summary

- A software life cycle model (or process model):
  - a descriptive and diagrammatic model of software life cycle
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities
  - divides life cycle into phases.
- A fundamental necessity while developing any large software product:
  - Adoption of a software development life cycle model (software process model).