Programming Engineering

Course 10 – 9 May

Outline

- Recap…
  - Design Patterns (Creational Patterns, Structural Patterns, Behavioral Patterns)
- Behavioral Patterns
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

Recap

- GOF: Creational Patterns, Structural Patterns, Behavioral Patterns
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

CP

- **Abstract Factory** – computer components
- **Builder** – children meal
- **Factory Method** – Hello <Mr/Ms>
- **Prototype** – Cell division
- **Singleton** – server log files

SP

- **Adapter** – socket-plug
- **Bridge** – drawing API
- **Composite** – employee hierarchy
- **Decorator** – Christmas tree
- **Façade** – store keeper
- **Flyweight** – FontData
- **Proxy** – ATM access

Behavioral Patterns

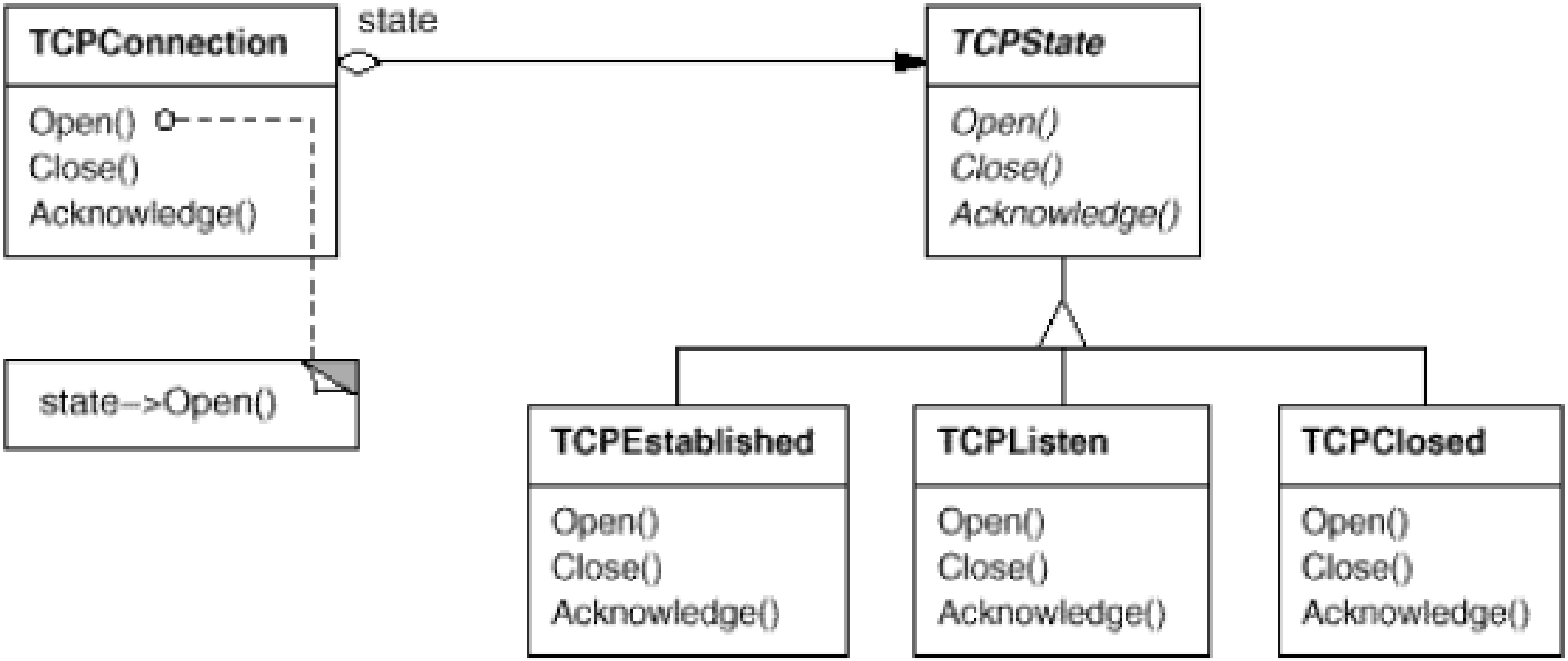
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

State

- **Intent** – Allow an object to alter its behavior when its internal state changes
- **Also Known As** – Objects for States
- **Motivation** – Consider a class TCPConnection that represents a network connection. A TCPConnection object can be in one of several different states: Established, Listening, Closed. When a TCPConnection object receives requests from other objects, it responds differently depending on its current state

State – Idea

- The key idea in this pattern is to introduce an abstract class called TCPState to represent the states of the network connection.



State – Applicability

- Use the State pattern in either of the following cases:
  - An object's behavior depends on its state
  - Operations have large, multipart conditional statements that depend on the object's state

State – The Good, The Bad …

- Simplifies the code of the objects by removing lots of if statements
- Preserves S and O (from SOLID)
- Leads to large and unnecessary overhead for objects with few states or whose states rarely change

Strategy

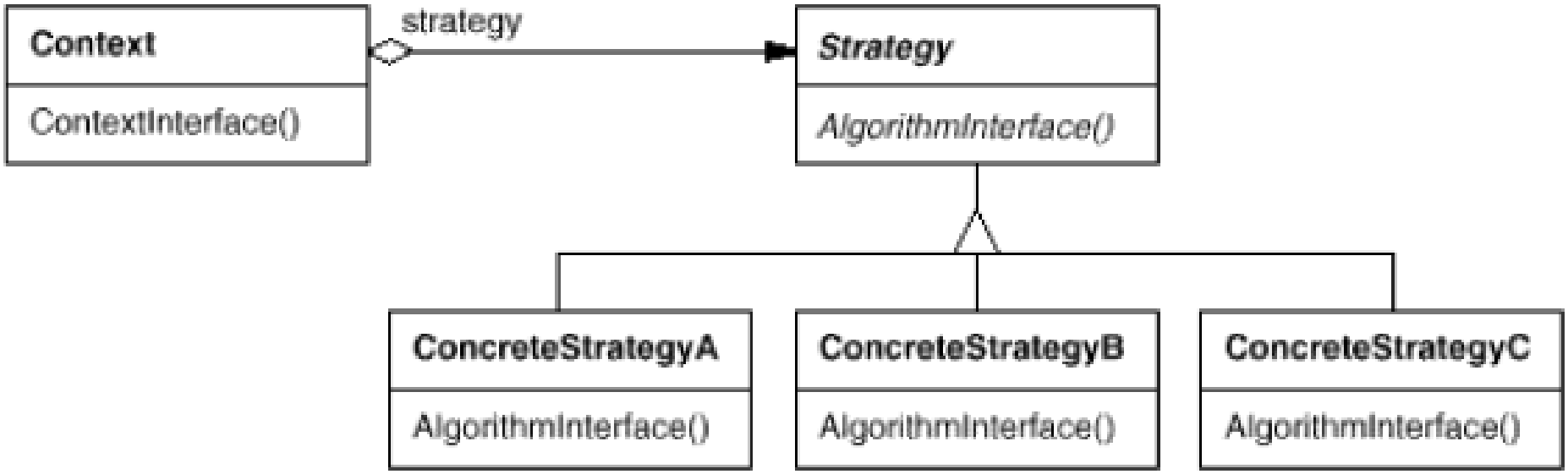
- **Intent** – Define a family of algorithms, encapsulate each one, and make them inter changeable

- **Also Known As** – Policy

- **Motivation** – Many algorithms exist for breaking a stream of text into lines. Hard-wiring all such algorithms into the classes that require them isn't desirable for several reasons

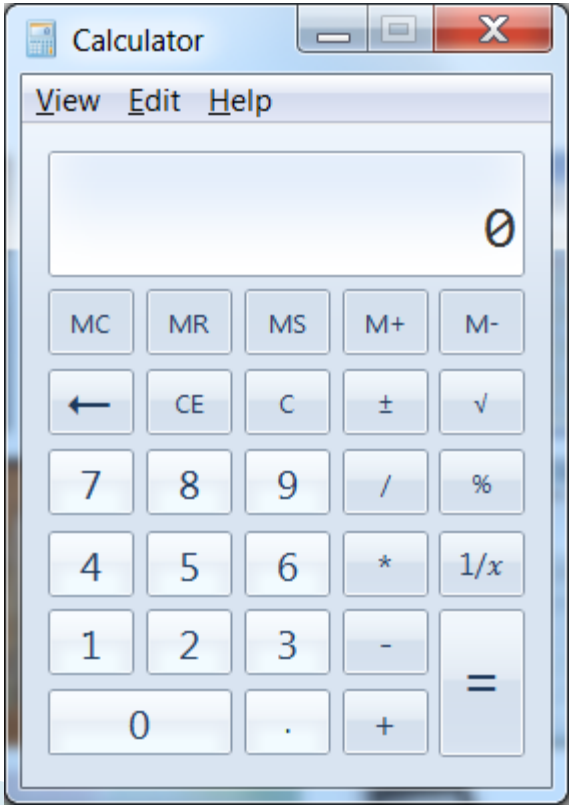
Strategy – Structure

- With Strategy pattern, we can define classes that encapsulate different line breaking algorithms



Strategy – Example

- In the strategy pattern algorithms can be selected at runtime.
- A standard calculator that implements basic operations: +, −, *



Strategy – Java 1

```
interface Strategy {
   int execute(int a, int b);
}
class ConcreteStrategyAdd implements Strategy {
   public int execute(int a, int b) {
         System.out.println("Called ConcreteStrategyA's execute()");
         return (a + b);
}}
class ConcreteStrategySub implements Strategy {
   public int execute(int a, int b) {
         System.out.println("Called ConcreteStrategyB's execute()");
         return (a - b);
}}
class ConcreteStrategyMul implements Strategy {
   public int execute(int a, int b) {
         System.out.println("Called ConcreteStrategyC's execute()");
         return a * b;
} }
```

Strategy – Java 2

```
class Context {
   Strategy strategy;
   public Context(Strategy strategy) {
        this.strategy = strategy;
   }
   public int execute(int a, int b) {
        return this.strategy.execute(a, b);
   }
}
```

Strategy – Java 3

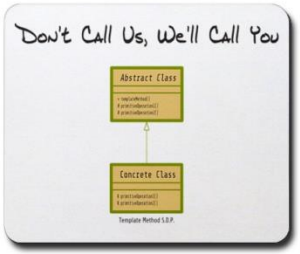
```
class StrategyExample {

   public static void main(String[] args) {
    Context context;
    context = new Context(new ConcreteStrategyAdd());
    int resultA = context.execute(3,4);
    context = new Context(new ConcreteStrategySub());
    int resultB = context.execute(3,4);
    context = new Context(new ConcreteStrategyMul());
    int resultC = context.execute(3,4);
   }
}
```

Strategy – The Good, The Bad …

- Can swap algorithms at runtime
- Replaces inheritance with composition
- Isolates the implementation of algorithm from the classes using those algorithm
- If you only need few algorithms, the extra complication of code is not useful
- Some functional languages can achieve the same effect by using anonymous functions, and require less code
- Users need to understand the differences between implementations to use them properly

Template Method



- **Intent – Define the skeleton of an algorithm** in an operation, deferring some steps to subclasses
- **Motivation** – Consider an application framework that provides Application and Document classes.
- The Application class is responsible for opening existing documents stored in an external format, such as a file. A Document object represents the information in a document once it's read from the file

Template Method – Example

- The template pattern is often referred to as the **Hollywood Principle**: "*Don't call us, we'll call you*." Using this principle, the template method in a parent class controls the overall process by calling subclass methods as required

- This is shown in several games in which players play against the others, but only one is playing at a given time

Template Method – Java 1

```
abstract class Game {
   protected int playersCount;
   abstract void initializeGame();
   abstract void makePlay(int player);
   abstract boolean endOfGame();
   abstract void printWinner();

   final void playOneGame(int playersCount) {
        this.playersCount = playersCount;
        initializeGame(); int j = 0;
        while (!endOfGame()) {
            makePlay(j); j = (j + 1) % playersCount; }
         printWinner();
   }
}
```

Template Method – Java 2

```
class Monopoly extends Game {
   // Implementation of necessary concrete methods

   void initializeGame() { // ... }
   void makePlay(int player) { // ... }
   boolean endOfGame() { // ... }
   void printWinner() { // ... }

   // Specific declarations for the Monopoly game.
}

class Chess extends Game {
...}
```

Template Method – The Good, The Bad …

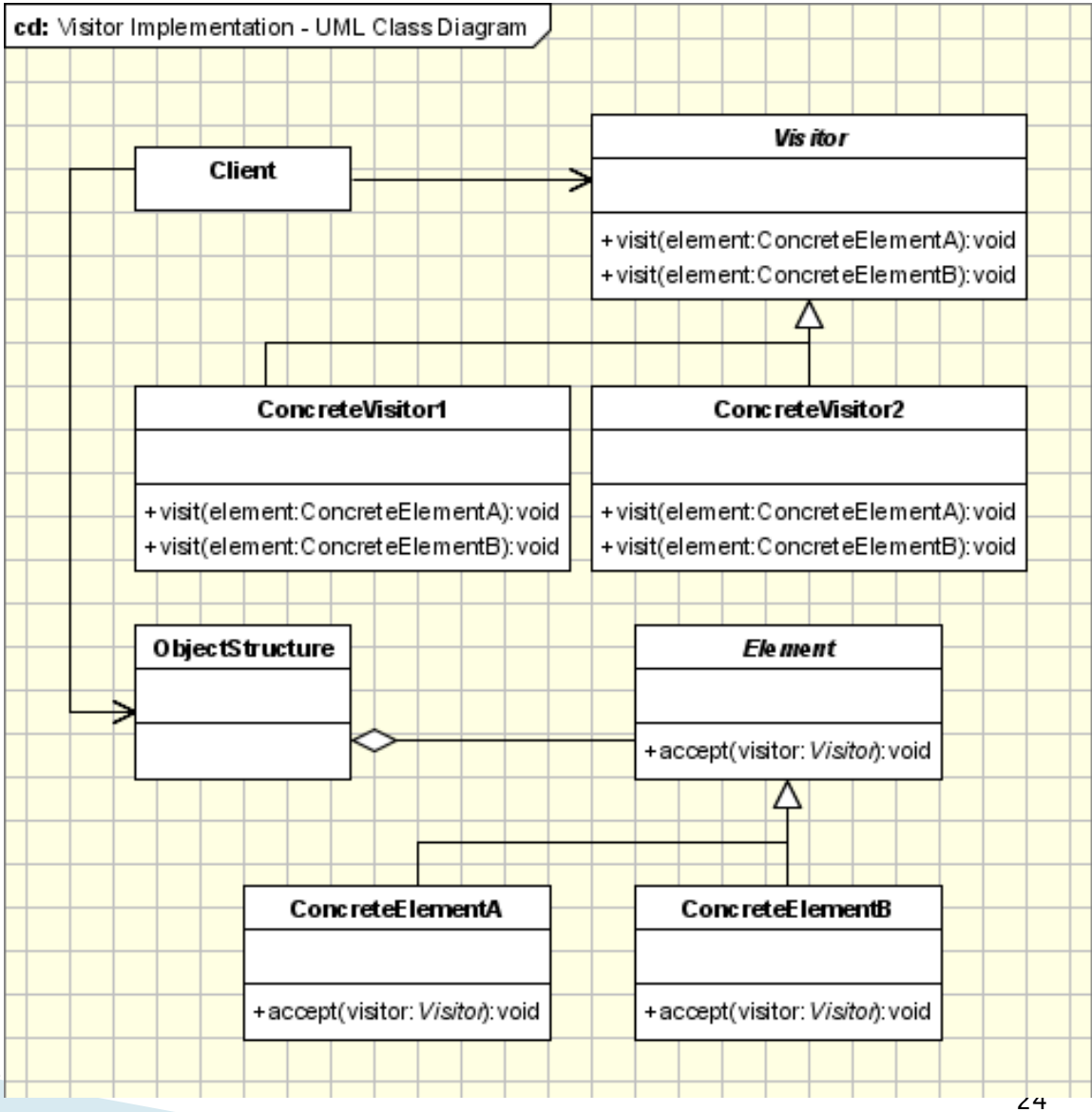
- Clients can choose to override only some parts of the algorithm, and are less affected by changes to other segments
- Duplicate code can be sent to a superclass
- If you override some part of the algorithm, it can lead to breaking Liskov substitution
- The skeleton of the algorithm may not suit some clients
- The more elements in the template, the more difficult it is to manage

Visitor

- **Intent** – Represent an operation to be performed on the elements of an object structure. **Visitor lets you define a new operation without changing the classes** of the elements on which it operates.
- **Motivation** – Collections are data types widely used in object oriented programming. Often collections contain objects of different types and in those cases some operations have to be performed on all the collection elements without knowing the type

Visitor – Structure Example

- We want to create a reporting module in our application to make statistics about a group of customers
- The statistics should be very detailed so all the data related to the customer must be parsed
- All the entities involved in this hierarchy must accept a visitor so the CustomerGroup, Customer, Order and Item are visitable objects



Visitor– Applicability

- The visitor pattern is used when:
  - Similar operations have to be performed on objects of different types grouped in a structure
  - There are many distinct and unrelated operations needed to be performed
  - The object structure is not likely to be changed but is very probable to have new operations which have to be added

Visitor Pattern using Reflection

- Reflection can be used to overcome the main drawback of the visitor pattern
- When the standard implementation of visitor pattern is used the method to invoke is determined at runtime
- **Reflection is the mechanism used to determine the method to be called at compile-time**

Visitor – The Good, The Bad …

- A visitor can accumulate information about visited objects as it passes through the collection, allowing for more informed decisions for later visitations
- Preserves S and O (from SOLID)
- A visitor may not be able to access private or protected fields of visited objects
- Every time you add an extra class to the collection, all visitors must be updated

Other Types of DP 1

- **Concurrency Patterns** – deal with multi-threaded programming paradigm
  - **Single Threaded Execution** – Prevent concurrent calls to the method from resulting in concurrent executions of the method
  - **Scheduler** – Control the order in which threads are scheduled to execute single threaded code using an object that explicitly sequences waiting threads
  - **Producer-Consumer** – Coordinate the asynchronous production and consumption of information or objects

Other Types of DP 2

- **Testing Patterns 1**
  - **Black Box Testing** – Ensure that software satisfies requirements
  - **White Box Testing** – Design a suite of test cases to exhaustively test software by testing it in all meaningful situations
  - **Unit Testing** – Test individual classes
  - **Integration Testing** – Test individually developed classes together for the first time
  - **System Testing** – Test a program as a whole entity

Other Types of DP 3

- **Testing Patterns 2**
  - **Regression Testing** – Keep track of the outcomes of testing software with a suite of tests over time
  - **Acceptance Testing** – Is done to ensure that delivered software meets the needs of the customer or organization that the software was developed for
  - **Clean Room Testing** – People designing software should not discuss specifications or their implementation with people designing tests for the software

Other Types of DP 4

- **Distributed Architecture Patterns**
  - **Mobile Agent** – An object needs to access very large volume of remote data => move the object to the data
  - **Demilitarized Zone** – You don't want hackers to be able to gain access to servers
  - **Object Replication** – You need to improve the throughput or availability of a distributed computation

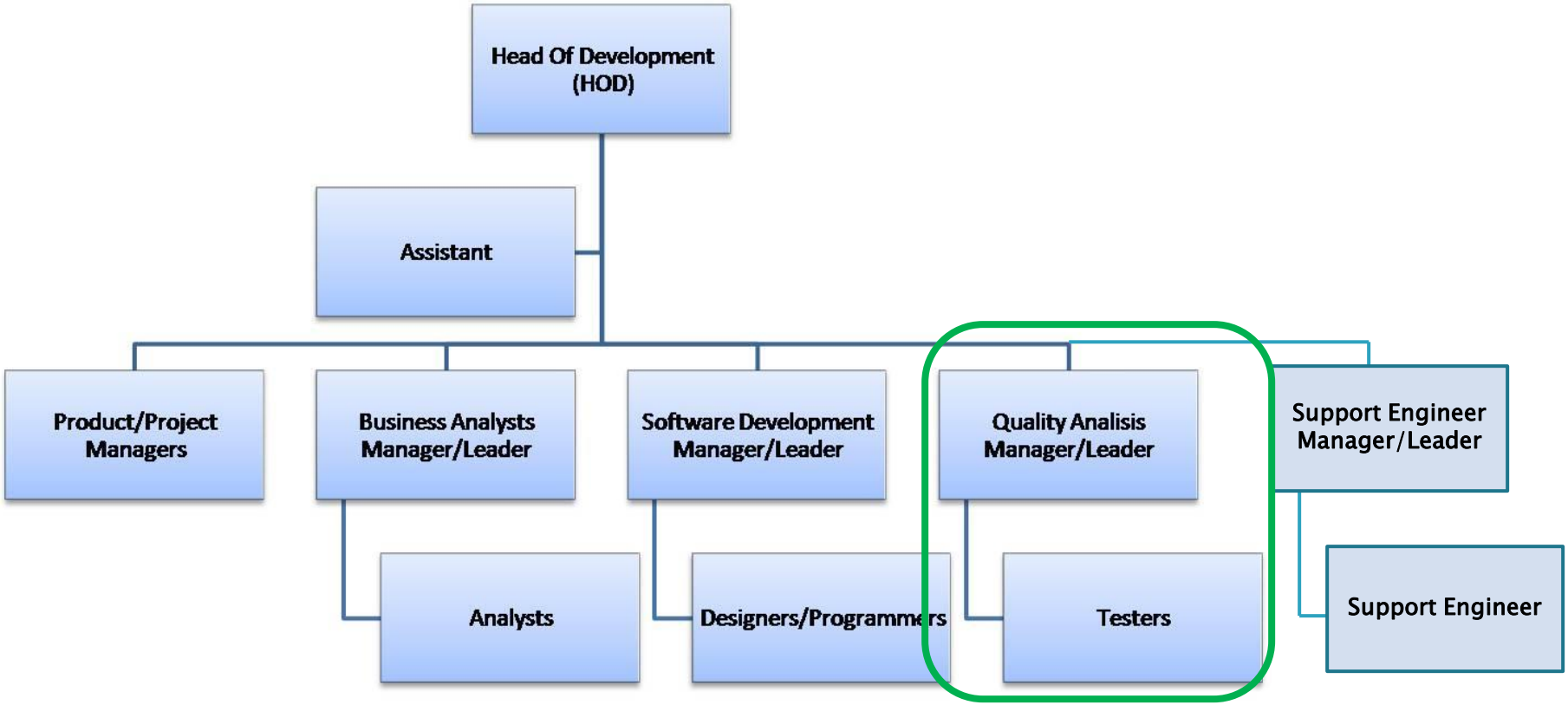
Other Classes of DP

- **Transaction patterns** – Ensure that a transaction will never have any unexpected or inconsistent outcome. Design and implement transactions correctly and with a minimum of effort
- **Distributed computing patterns**
- **Temporal patterns** – for distributed applications to function correctly, the clocks on the computers they run on must be synchronized. You may need to access pervious or future states of an object. The values of an object's attributes may change over time
- **Database patterns**

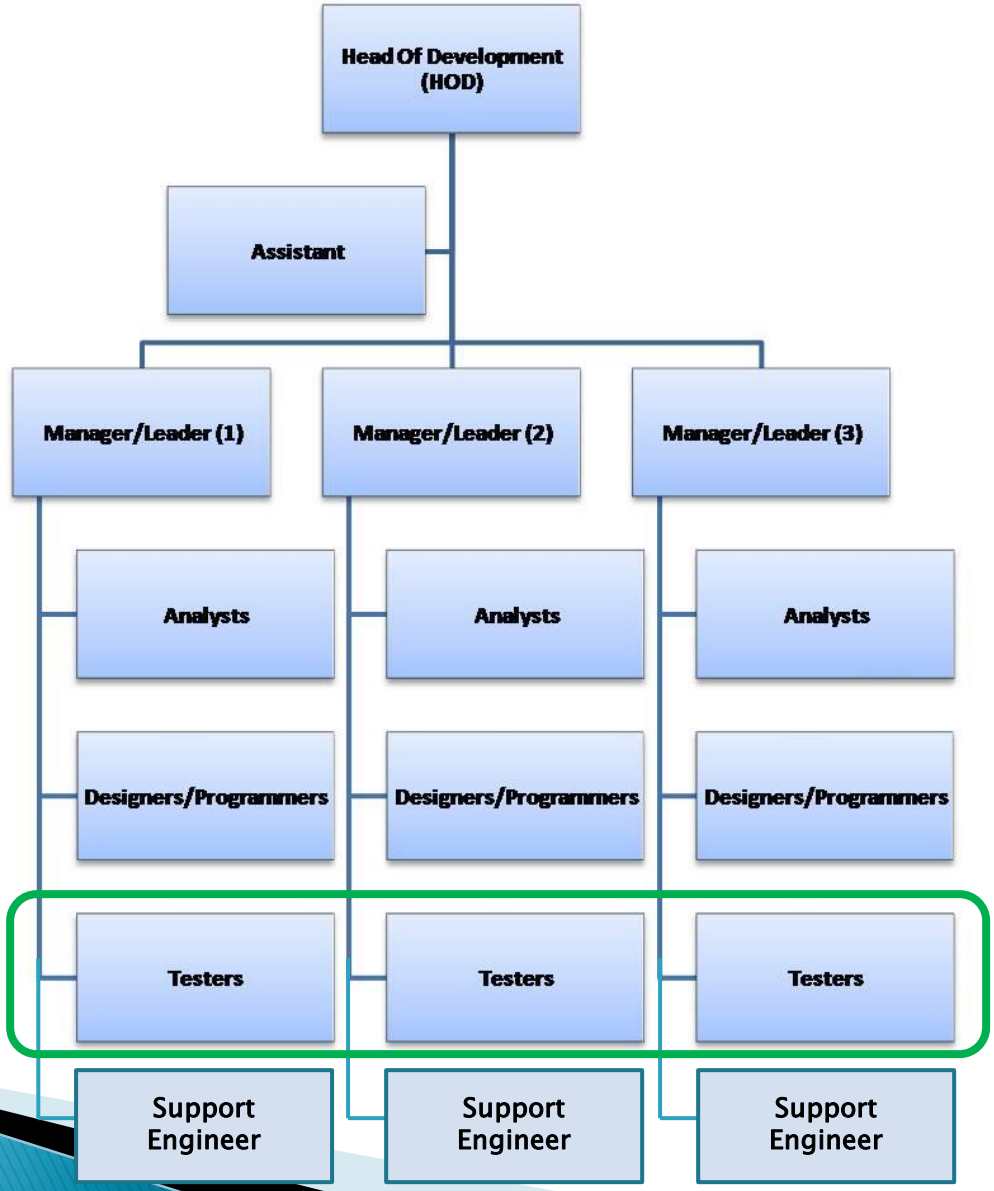
Quality Assurance

- Quality Assurance
  - Software Testing
  - Testing Methodologies
  - Testing process
  - Manual Testing vs. Automatic Testing

Hierarchy of an IT Company – Compartments



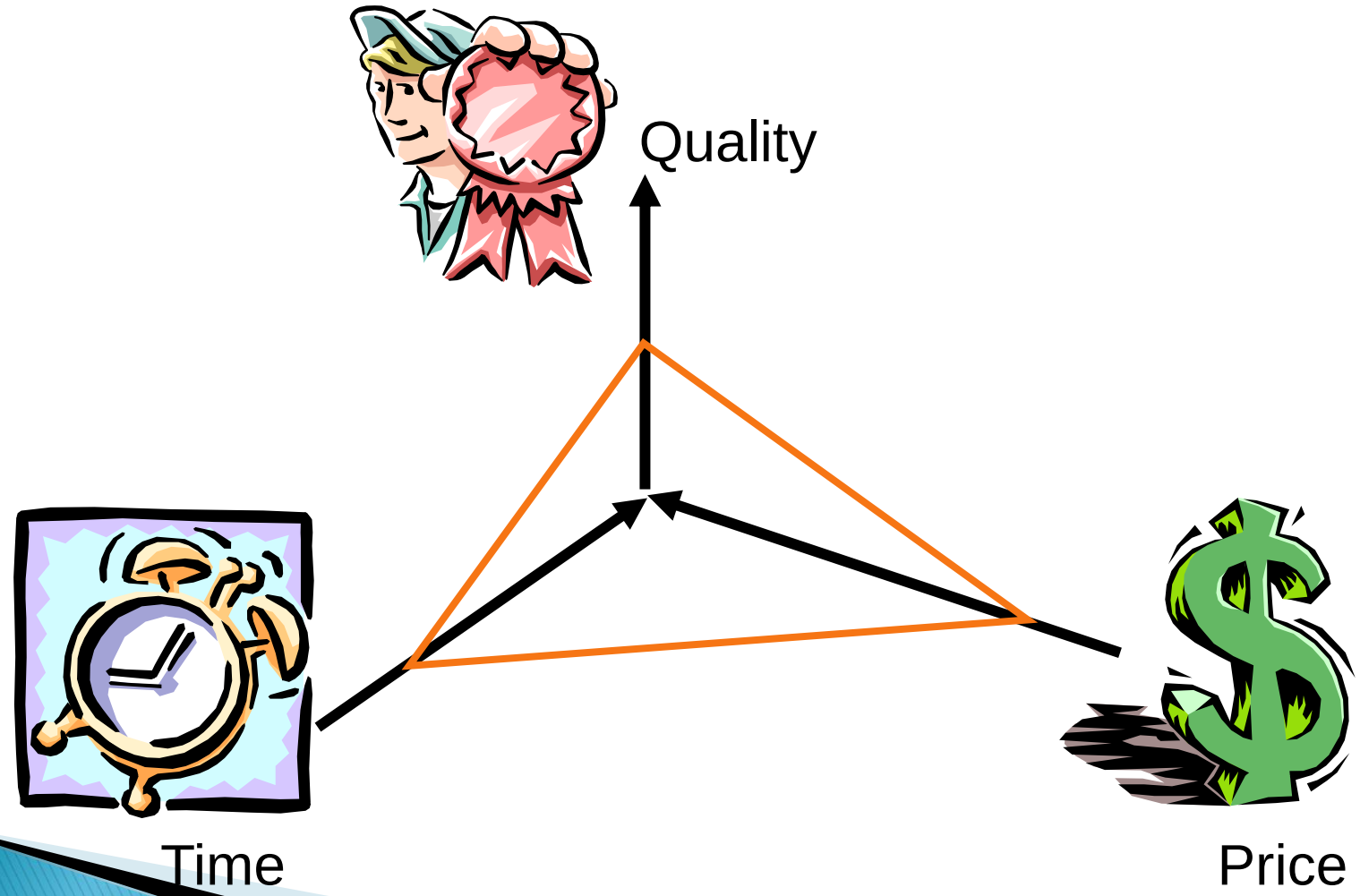
Hierarchy of an IT Company – Projects



Quality Assurance

- **Planned and systematic production processes** that provide confidence in a product's suitability for its intended purpose.
- A set of activities intended to ensure that **products satisfy customer requirements**
- *QA cannot absolutely guarantee the production of quality products but makes this more likely*
- Two key principles of QA:
  - "fit for purpose" – the product should be suitable for the intended purpose, and
  - "right first time" – mistakes should be eliminated

Quality Assurance Dilemma



Quality Assurance – Definition

"The process of exercising or evaluating a system by manual or automated means to verify that it satisfies specified <u>requirements</u> or to identify differences between expected and actual results."

(IEEE Standard Glossary, 1983)

Software Quality Assurance

- (SQA) consists of a means of **monitoring the software engineering processes and methods used to ensure quality**
- May include ensuring conformance to one or more standards, such as ISO 9000 or CMMI

Software Quality Assurance

- SQA encompasses the entire software development process, which includes processes such as
  - *software design,*
  - *coding,*
  - *source code control,*
  - *code reviews,*
  - *testing,*
  - *change management,*
  - *configuration management, and*
  - *release management*

ISO 9000

- ISO 9000 is a family of standards for quality management systems
- Some of the requirements in ISO 9001 (from ISO 9000 family) include
  - a set of procedures;
  - monitoring processes;
  - keeping adequate records;
  - checking output for defects;
  - regularly reviewing individual processes;
  - facilitating continual improvement

Software Quality Assurance

- Quality assurance is concerned with prevention
  - Defect Prevention
  - Processes
  - Continuous improvement of this processes
- Quality control is concerned with correcting
  - Software testing is a subset of Quality control

Image credit test-institute.org

Software Testing – Introduction

- An empirical investigation conducted to provide information about the *quality of the product or service under test, with respect to the context in which it is intended to operate*.
- Allow the business to *appreciate and understand* the risks at implementation of the software
- Test techniques include the process of executing a program or application with **the intent of finding software bugs**
- The process of **validating and verifying** that a software program/application/product meets the business and technical requirements that guided its design and development

Software Testing – When?

- Can be implemented at any time in the development process
- However the most test effort is employed after the requirements have been defined and coding process has been completed

Software Testing

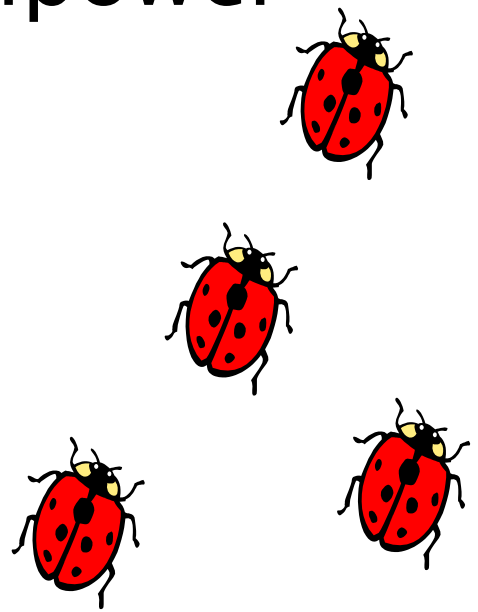
- Software Testing is NOT a phase
- It is integrated in all phases of software development
- Each development step has an attached testing documentation

What is the purpose of testing?

- We need not only to find bugs but also to prevent them (which is better)
- To know when to stop because effectiveness and economics of the process is essential.
- To know that not all system requires the same level of quality (mission critical against IT).
- Testing is not only for the SOFTWARE it is for all DELIVERABLES

Why are there bugs in software?

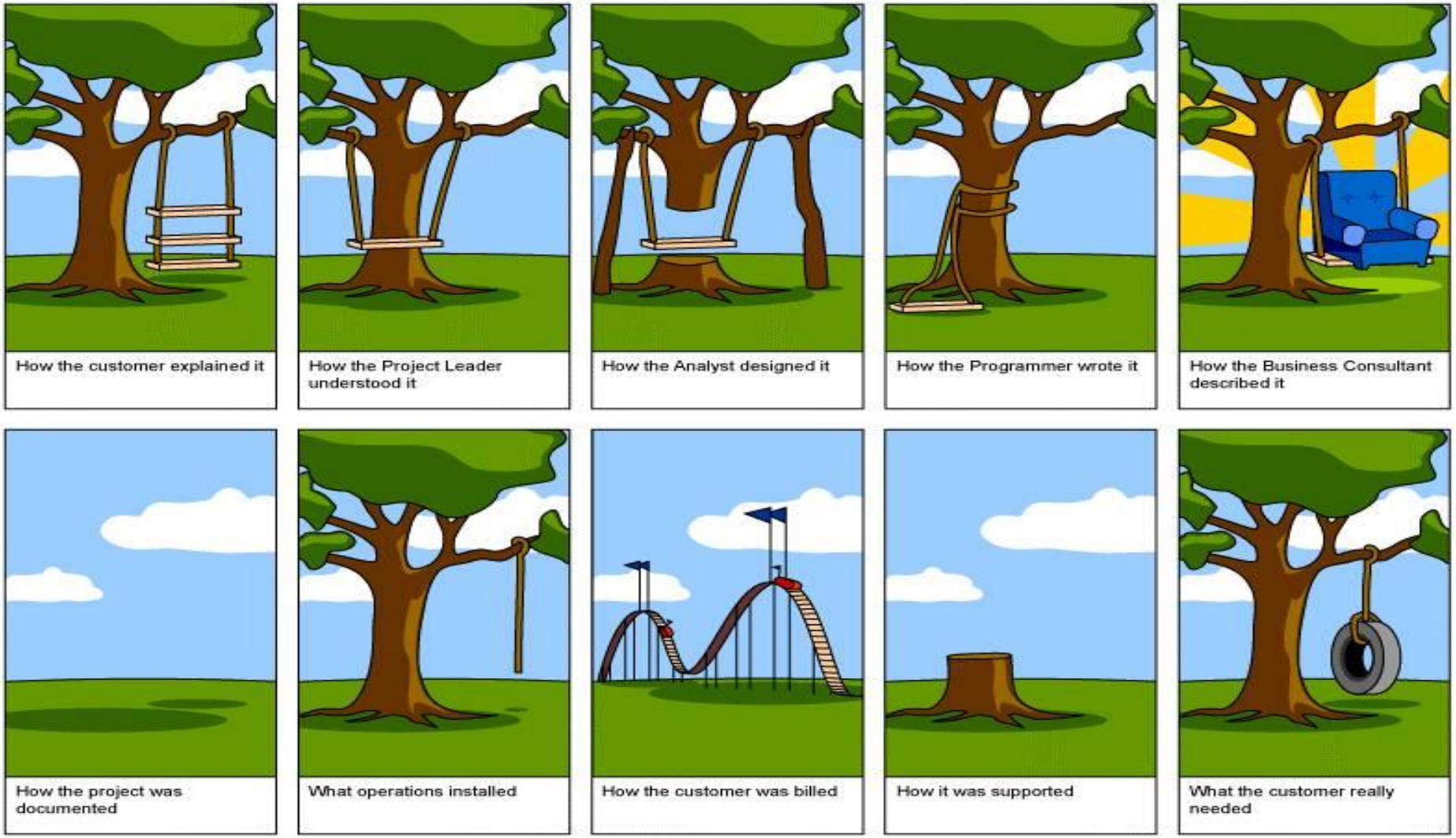
- Miscommunication
- Misunderstanding
- Low professional manpower
- Time pressures



Miscommunication

"Didn't you get my e-mail?"

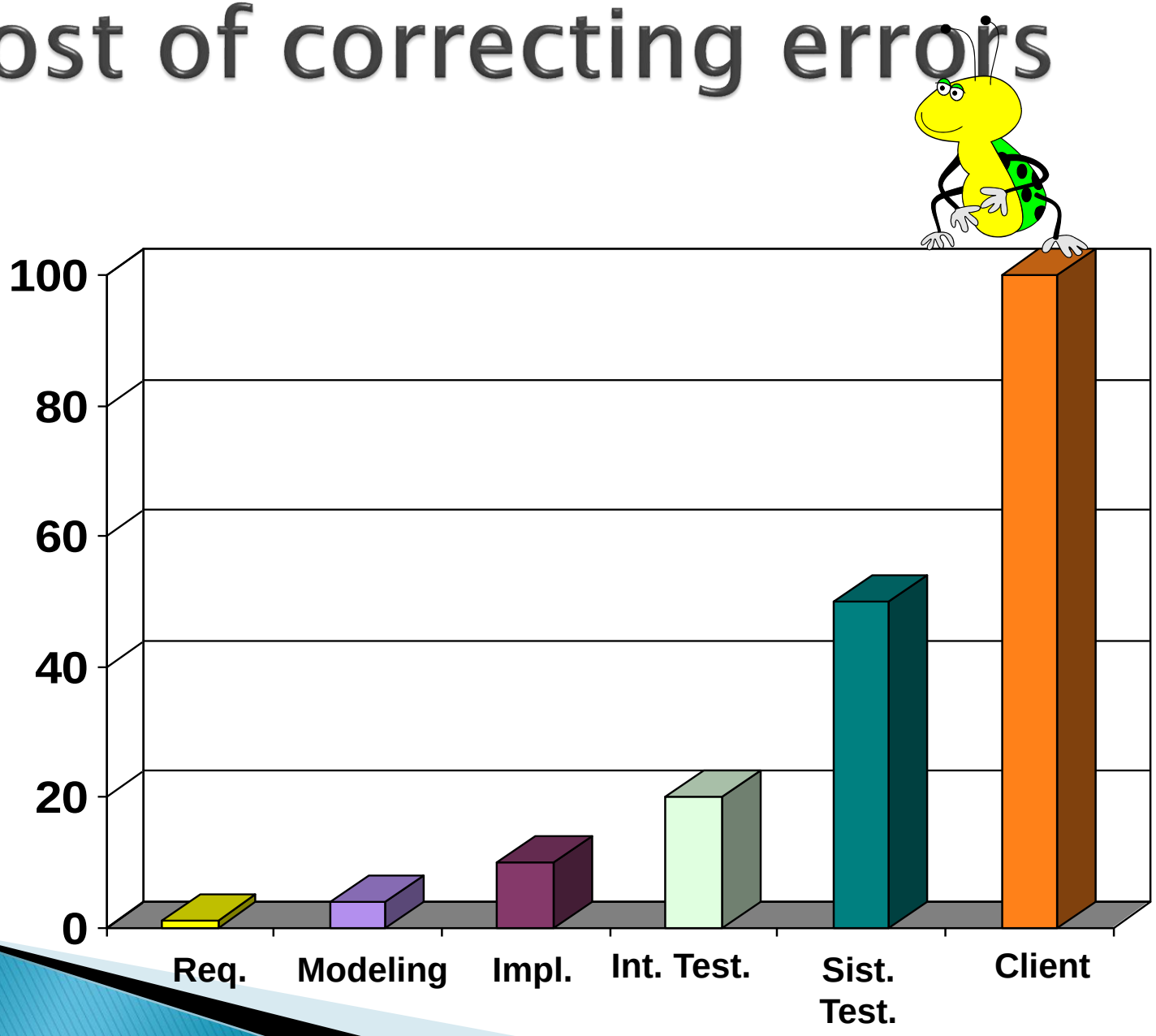
Misunderstandings



What genenerates most errors?

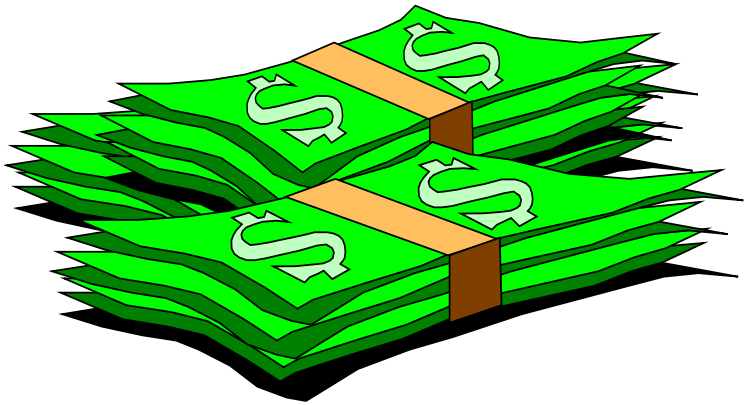
- Incorrect or incomplete requirements 50%
- Ambiguous or incomplete modeling 30%
- Programming errors 20%

Cost of correcting errors

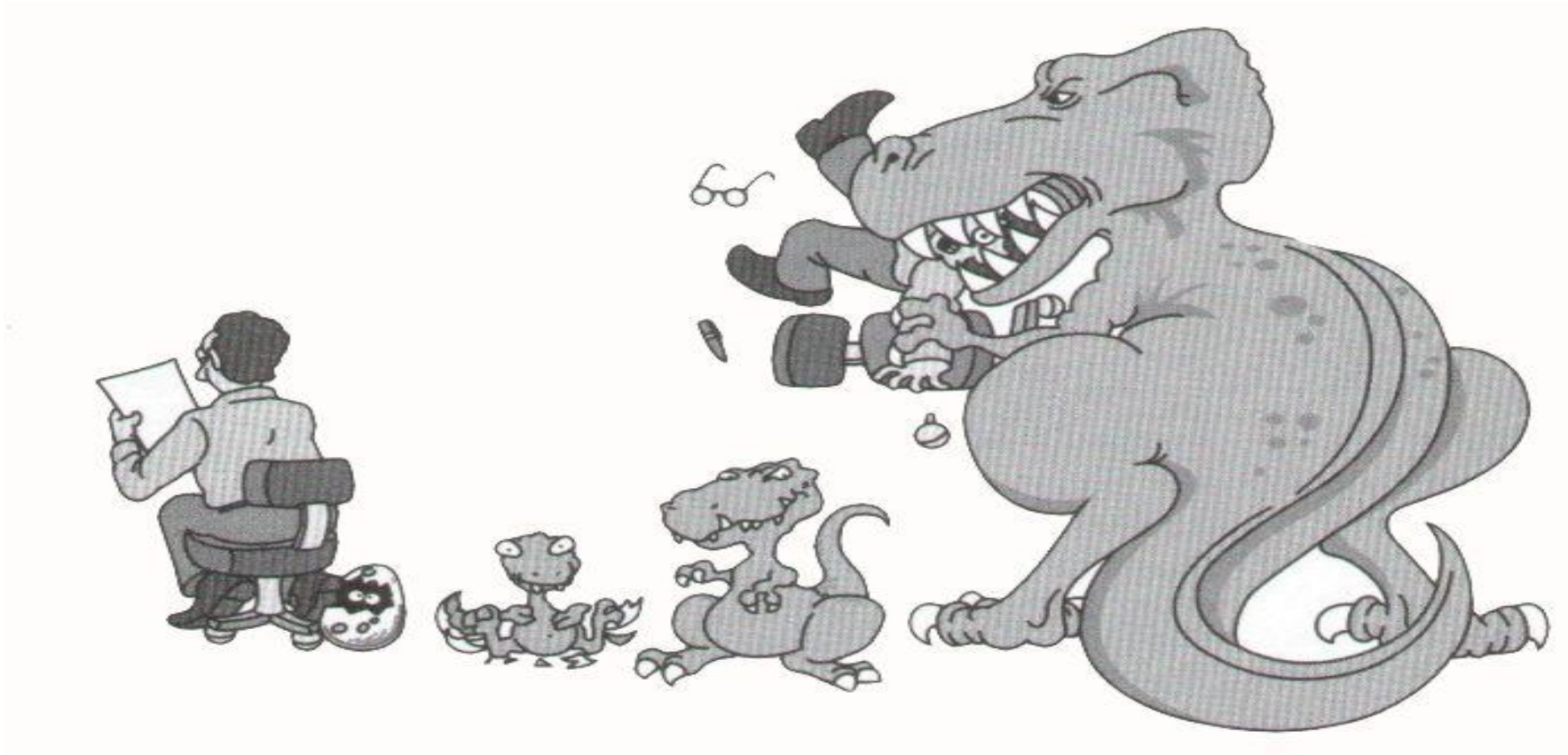


Note

Late error detection⇒ greater repair cost



Error must be repaired as soon as possible



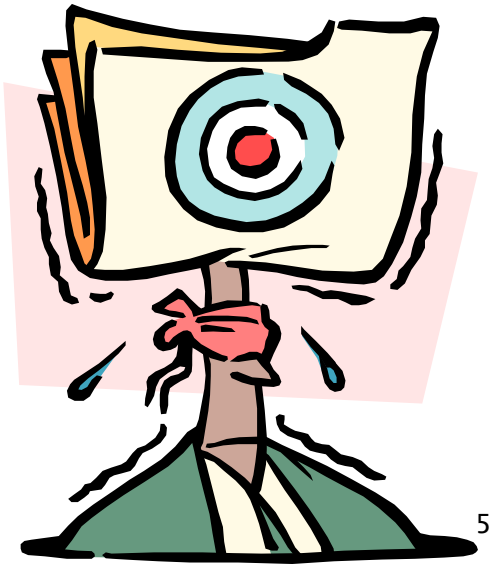
REQ. MODELING IMPLEM. CLIENT TESTING

Professional Testing

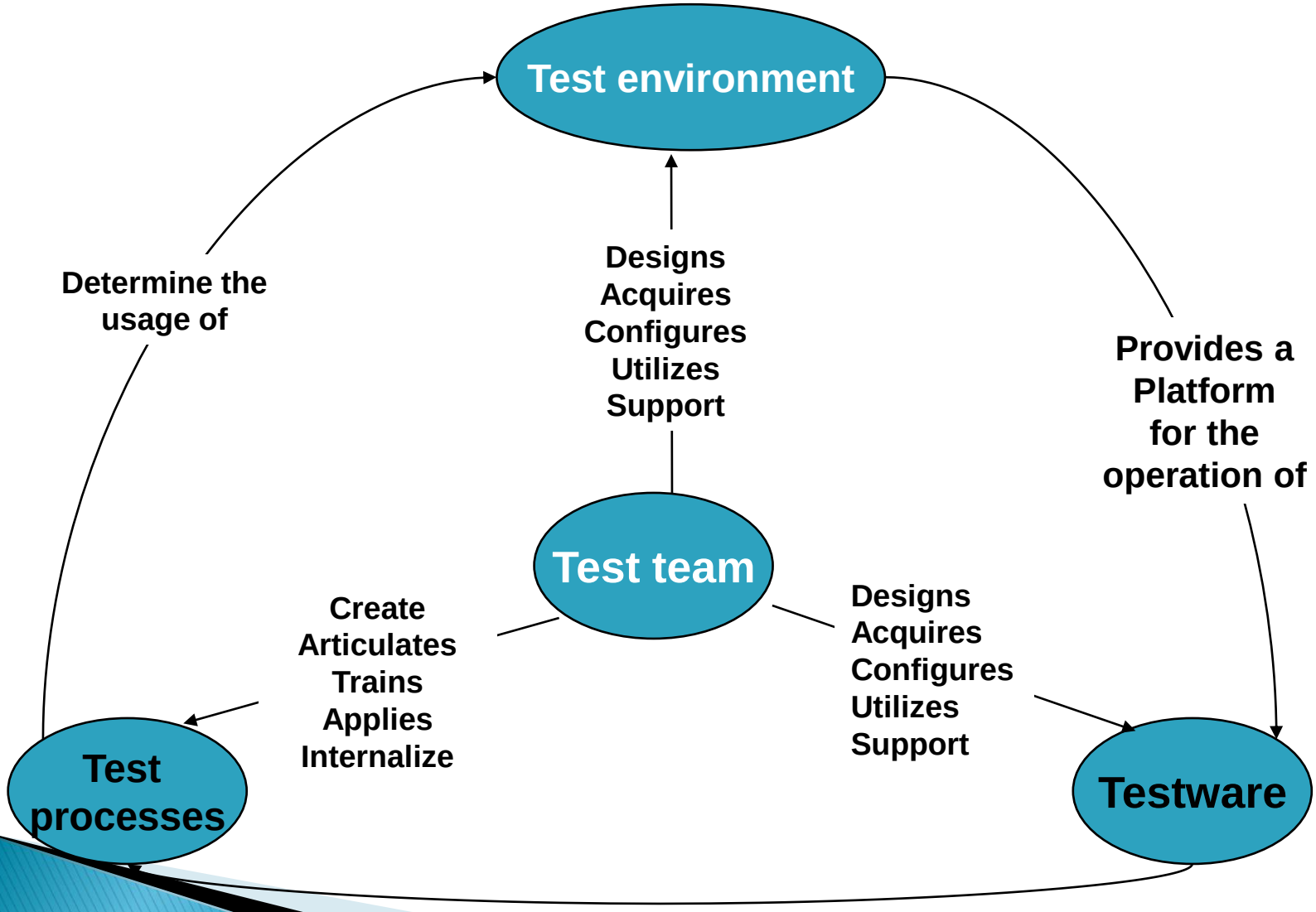
Professional testing means finding the least amount of test cases which will check the most amount of system features.

When does testing stop?

- Never
- When the number of errors found in a test cycle is lower than a set amount
- When no more critical faults are found
- When we run out of time



Schema of a Testing System



Testing Methodology

- Differences between testing and debugging
- Layers of testing
- Testing methods
- Testing content
- Manual vs Automatic Testing

Testing vs. Debugging

Testing

- Check compliance to requirements
- Normally carried out by an external and neutral party
- Is planned and controlled

Debugging

- Check validity of program sections
- Run by the developer
- Is a random process

Layers of testing

- Unit testing or debugging
- Module/Sub-System
- Integration
- System
- Acceptance

Unit Testing

- Testing a function, program screen, feature
- Run by programmers
- Predefined
- Results must be documented
- Input and Output simulators are used

Integration testing

- Testing of several modules at the same time
- Testing coexistence
- Run by programmers or testers
- Pre-planned testing
- Results must be documented

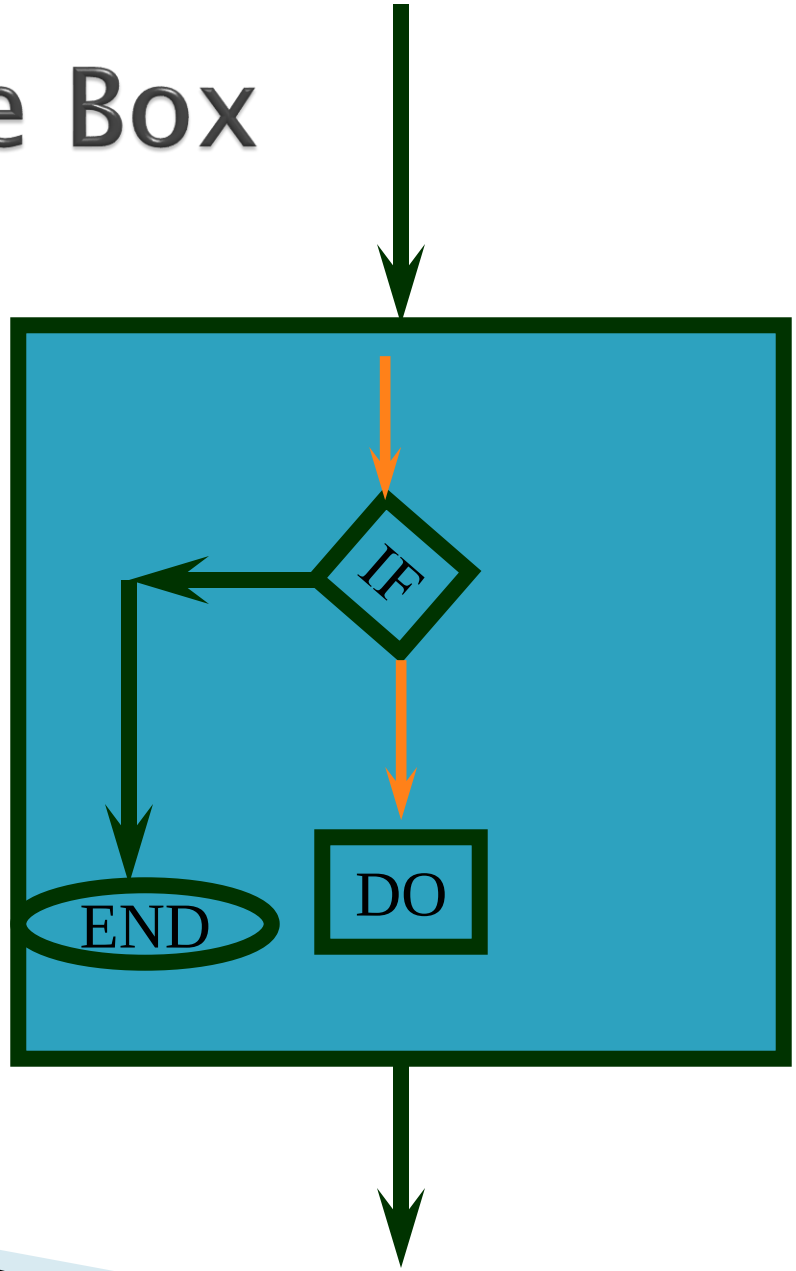
System Testing

- System testing of software or hardware is testing conducted **on a complete, integrated system** to evaluate the system's compliance with its specified requirements.
- System testing falls within the scope of **black box testing**
- System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

Testing methods

- White Box
- Black Box
- Gray Box
- Graphical user Interface Testing
- Acceptance Testing
- Regression Testing

White Box

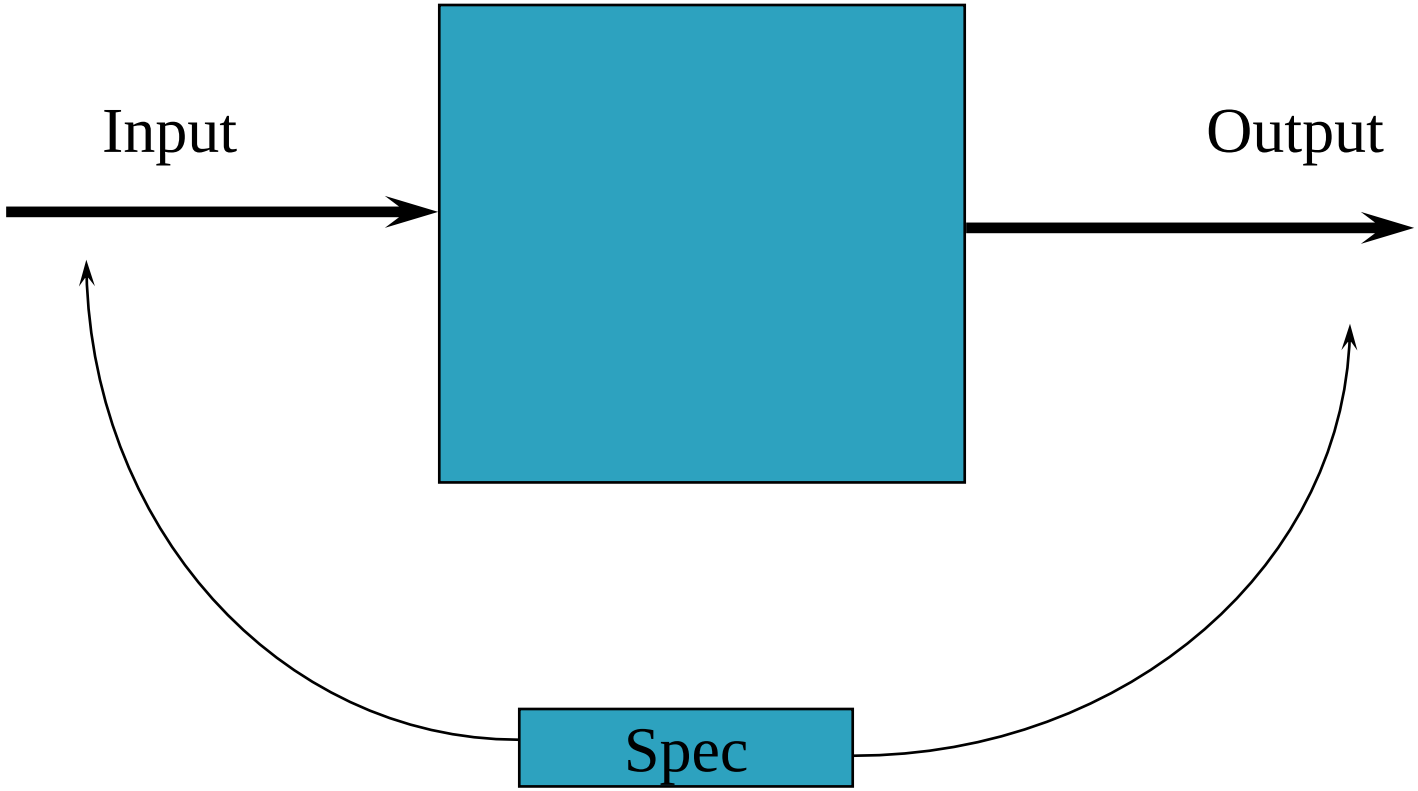


White Box (2)

- The tester has access to the internal data structures and algorithms

- Types of white box testing
  - api testing – Testing of the application using Public and Private APIs
  - code coverage – creating tests to satisfy some criteria of code coverage
  - fault injection methods
  - mutation testing methods
  - static testing – White box testing includes all static testing

Black Box



- "like a walk in a dark labyrinth without a flashlight,"

Black Box (2)

- Specification-based testing

- Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzzy testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

Grey Box

- This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level
- Manipulating input data and formatting output do not qualify as "grey-box," because the input and output are clearly outside of the "black-box" that we are calling "the software under test"

GUI Testing

- In computer science, GUI software testing is the process of testing a product that uses a graphical user interface, to ensure it meets its written specifications.
- The variety of errors found in GUI applications:
  - Data validation, Incorrect field defaults, Mandatory fields, not mandatory, Wrong fields retrieved by queries, Incorrect search criteria
  - Field order, Multiple database rows returned, single row expected
  - Currency of data on screens, Correct window modality?
  - Control state alignment with state of data in window?

Acceptance Testing

- **A black-box testing performed on a system** prior to its delivery
- In software development, acceptance testing by the system provider is often distinguished from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership.
- In such environments, acceptance testing performed by the customer is known as **user acceptance testing** (UAT).
- This is also known as **end-user testing**, site (acceptance) testing, or field (acceptance) testing.

Regression Testing

- Regression testing is **any type of software testing** which seeks to uncover software regressions.
- Such regressions occur whenever software functionality that was previously working correctly, stops working as intended.
- Typically regressions **occur as an unintended consequence of program changes**.
- Common methods of regression testing include **re-running previously run tests and checking whether previously fixed faults have re-emerged**

Bibliography

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software* (GangOfFour)

Links

- Structural Patterns: http://www.oodesign.com/structural-patterns/
- Gang-Of-Four: http://c2.com/cgi/wiki?GangOfFour, http://www.uml.org.cn/c%2B%2B/pdf/DesignPatterns.pdf
- Design Patterns Book: http://c2.com/cgi/wiki?DesignPatternsBook
- About Design Patterns: http://www.javacamp.org/designPattern/
- Design Patterns – Java companion: http://www.patterndepot.com/put/8/JavaPatterns.htm
- Java Design patterns: http://www.allapplabs.com/java_design_patterns/java_design_patterns.htm
- Overview of Design Patterns: http://www.mindspring.com/~mgrand/pattern_synopses.htm https://refactoring.guru/design-patterns