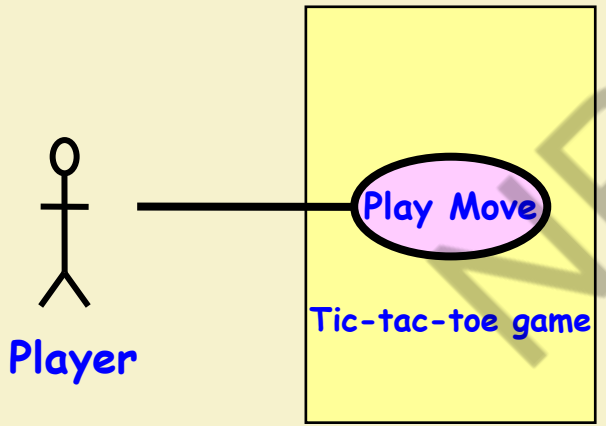
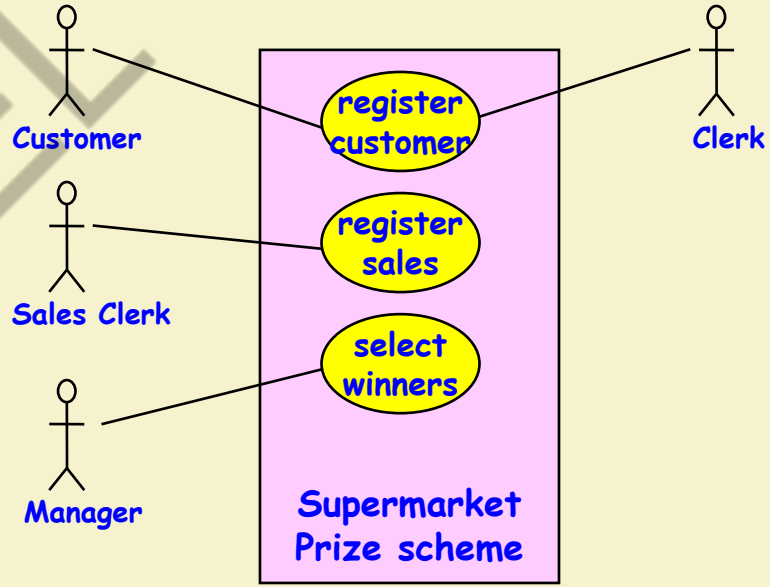
Domain Analysis

- Three types of classes are to be identified:
  - Boundary class (Actor-use case pair)
  - Controller class (One per use case)
  - Entity class (Noun analysis)

Identification of Boundary Objects

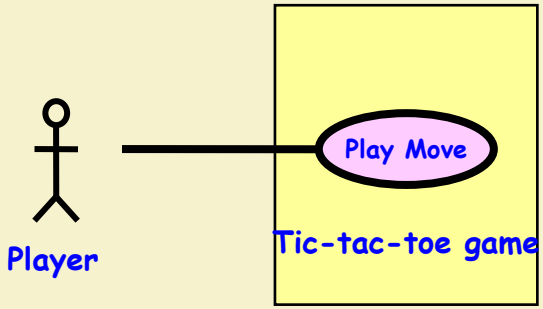
- Need one boundary object :
  - For every actor-use case pair

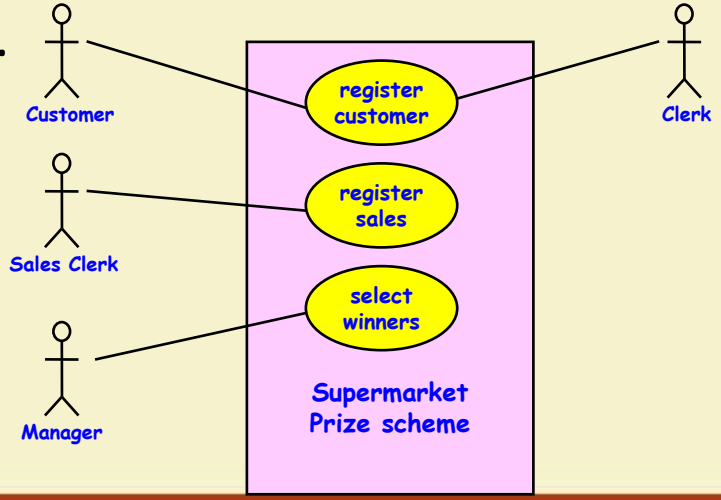




Identification of Controller Objects

- Examine the use case diagram:
  - **Add one controller class for each use case.**
  - Some controllers may need to be split into two or more controller classes if they get assigned too much responsibility.





Identification of Entity Objects by Noun Analysis

- Entity objects usually appear as nouns in the problem description.
- From the list of nouns, need to exclude:
  - **Users (e.g. accountant, librarian, etc)**
  - **Passive verbs (e.g. Acknowledgment)**
  - **Those with which you can not associate any data to store**
  - **Those with which you can not associate any methods**
- Surrogate users may need to exist as classes:
  - **Library member**

- Remember that a class represents a group (classification) of objects with the same behavior.
  - **We should therefore look for existence of similar objects during noun analysis**
- Even then, class names should be singular nouns:
  - Examples: **Book, Student, Member**

Identifying Classes

Noun Analysis: Example

A trading house maintains names and addresses of its regular customers. Each customer is assigned a unique customer identification number (CIN). As per current practice, when a customer places order, the accounts department first checks the credit-worthiness of the customer.

Identifying Classes by Noun Analysis

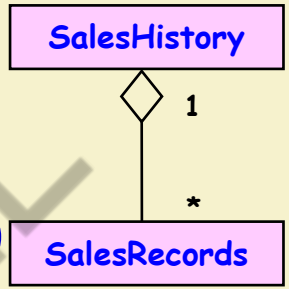
- A partial requirements document:

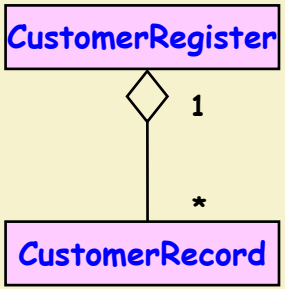
A trading house maintains names and addresses of its regular customers. Each customer is assigned a unique customer identification number (CIN). As per current practice, when a customer places order, The accounts department first checks the credit-worthiness of the customer.

- Not all nouns correspond to a class in the domain model

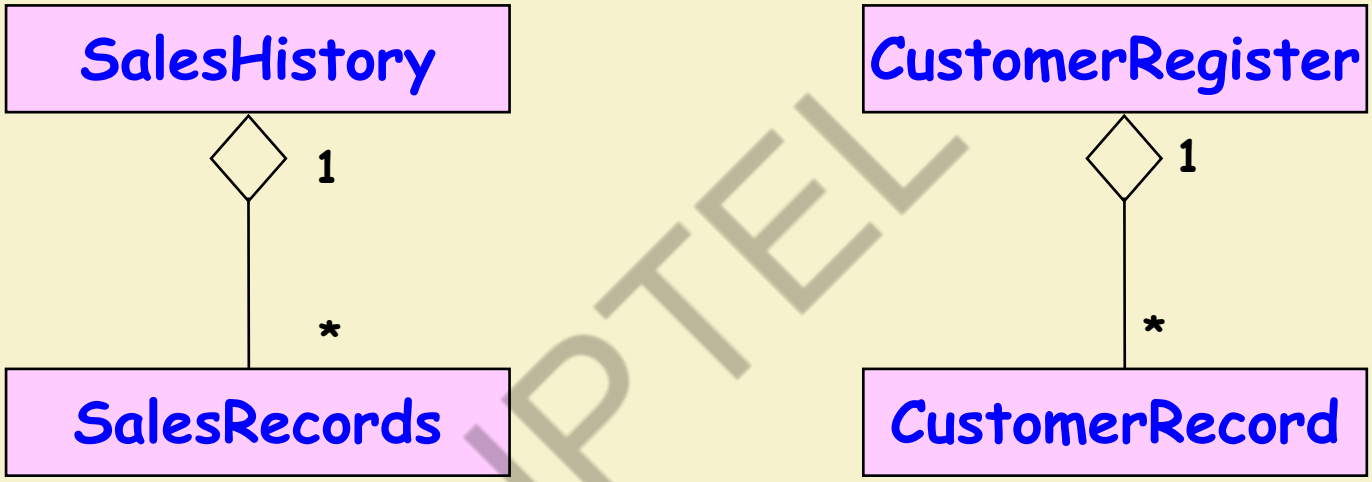
Identification of Entity Objects

- **Usually:**
  - Appear as data stores in DFD
  - Occur as group of objects that are aggregated
  - The aggregator corresponds to registers in physical world





Example 2: Initial Domain Model



Initial domain model

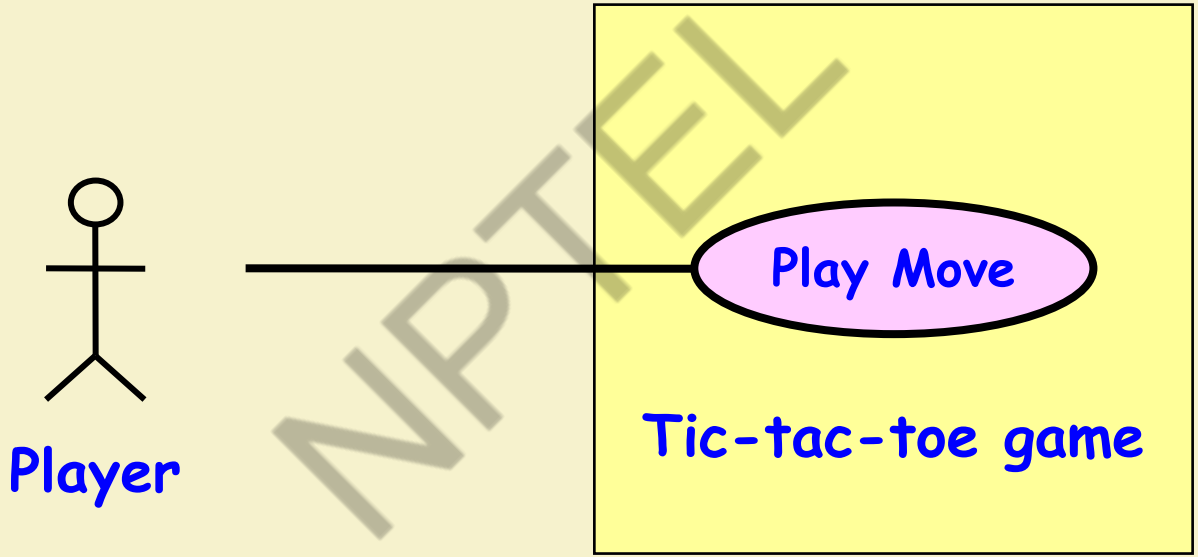
Example 1: Tic-Tac-Toe Computer Game

- A human player and the computer make alternate moves on a 3X3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.

Example 1: Tic-Tac-Toe Computer Game cont…

- As soon as either of the human player or the computer wins,
  - A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
  - And all the squares on the board are filled up,
  - Then the game is drawn.
- The computer always tries to win a game.

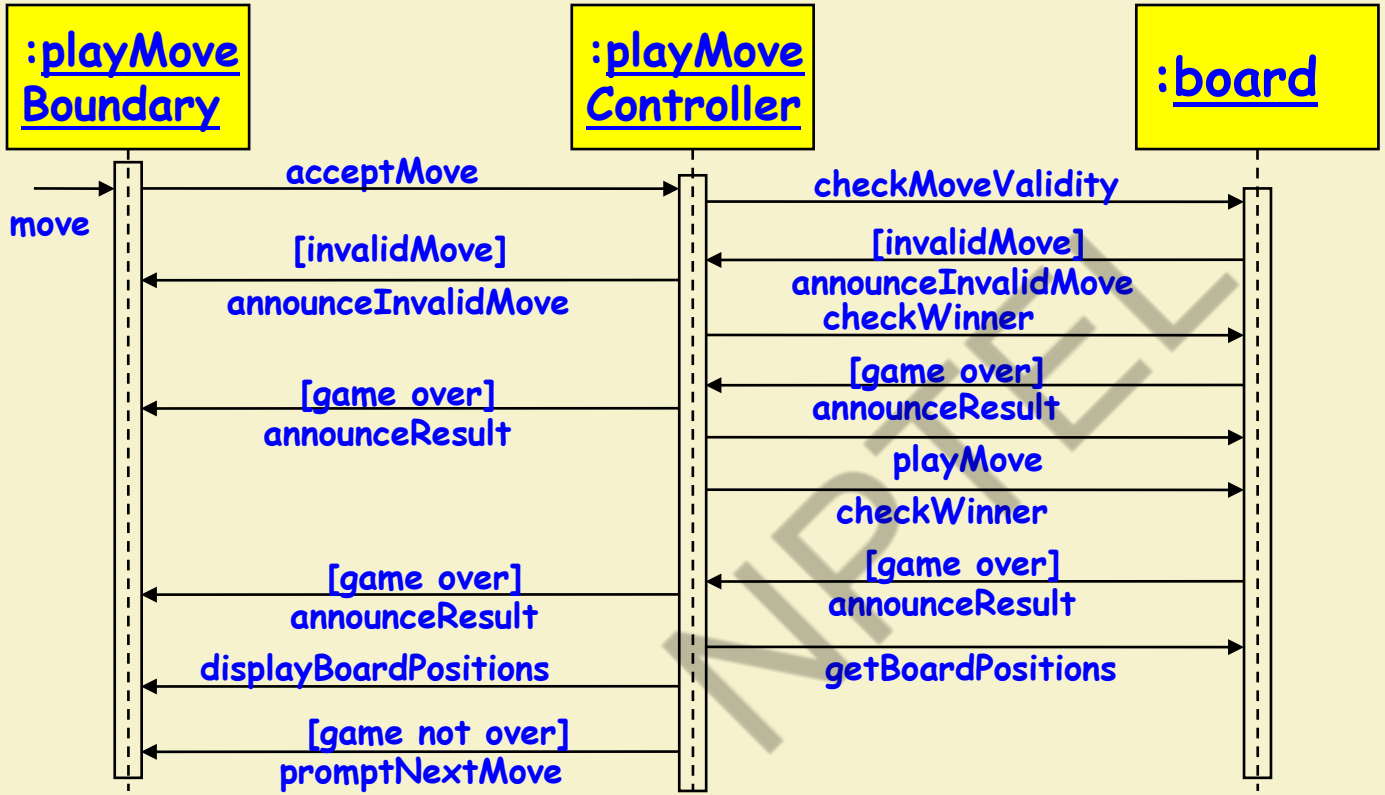
Example 1: Tic-Tac-Tie
Use Case Model



Example 1: Initial and Refined Domain Model

Initial domain model

Refined domain model



Example 1: Sequence Diagram: play move use case

CRC Card

- **Used to assign responsibilities (methods) to classes.**
- Complex use cases:
  - Realized through collaborative actions of dozens of classes.
  - Without CRC cards, it becomes difficult to determine which class should have what responsibility.

Class-Responsibility-Collaborator(CRC) Cards

- Pioneered by Ward Cunningham and Kent Beck.
- Index cards prepared one each per class.
- Contains columns for:
  - Class responsibility
  - Collaborating objects

Class name	
Responsibility	Collaborator

CRC Cards Cont…

- **Systematize development of interaction diagram for complex use cases.**
- Team members participate to determine:
  - –The responsibility of classes involved during a use case execution

CRC Cards Cont...

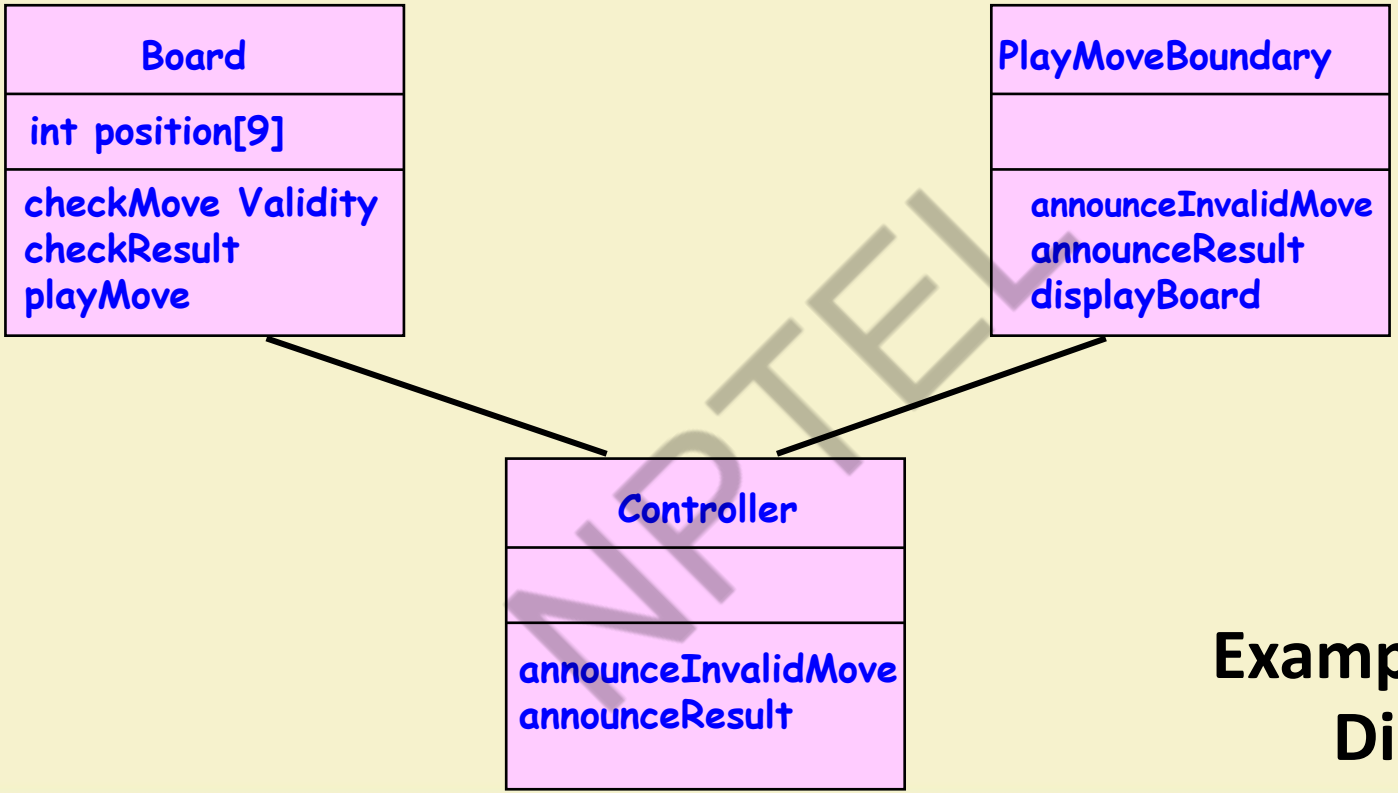
- **Responsibility:**
  - Method to be supported by the class.
- **Collaborator:**
  - Class whose service (method) would have to be invoked

Class name	
Responsibility	Collaborator

An Example: CRC Card for the BookRegister class

Using CRC Cards

- After developing a set of CRC cards:
  - Run structured walkthrough scenarios
- Walkthrough of a scenario :
  - A class is responsible to perform some responsibilities
  - It may then pass control to a collaborator -- another class
  - You may discover missing responsibilities and classes



Example 1: Class Diagram

Example 2: Supermarket Prize Scheme

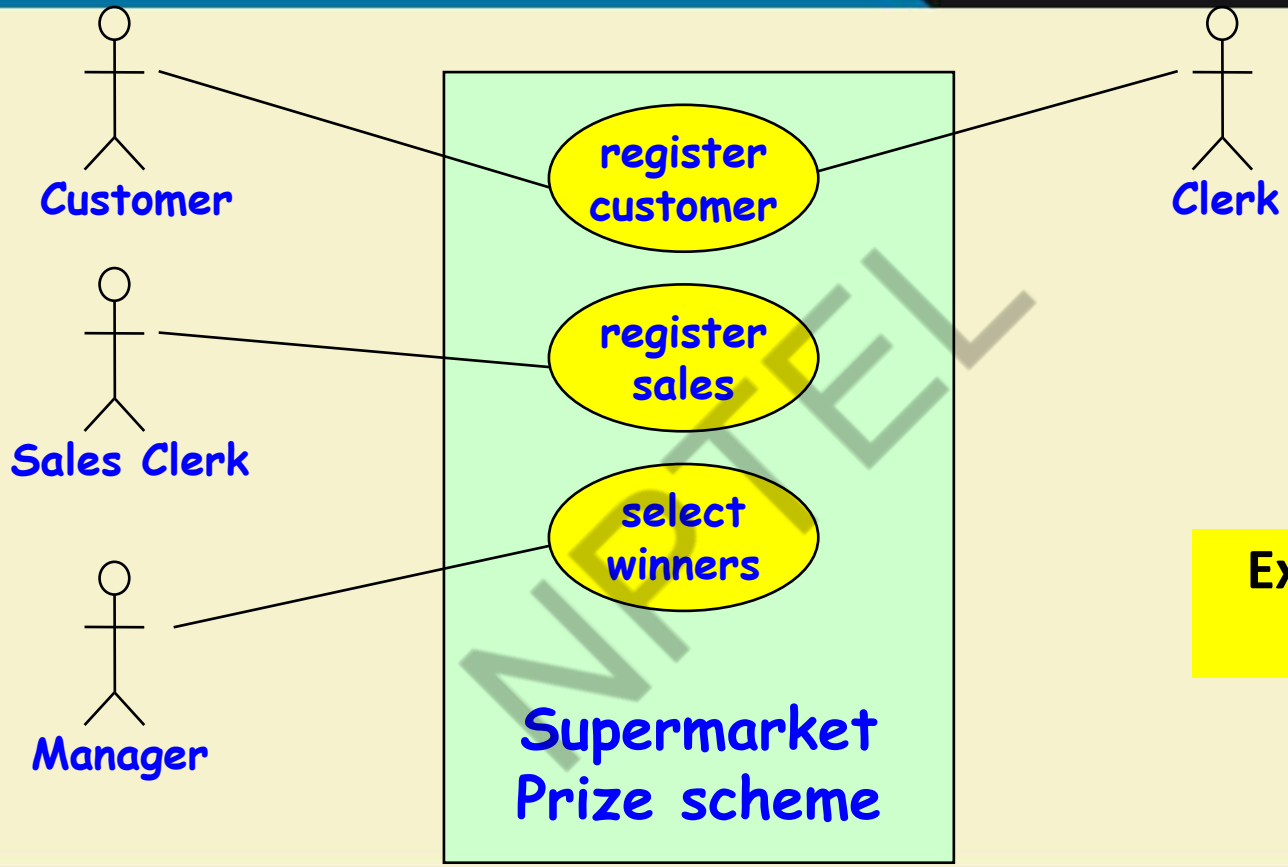
- Supermarket needs to develop software to encourage regular customers.
- Customer needs to supply his:
  - Residence address, telephone number, and the driving licence number.
- Each customer who registers is:
  - Assigned a unique customer number (CN) by the computer.

Example 2: Supermarket Prize Scheme

- A customer can present his CN to the staff when he makes any purchase.
- The value of his purchase is credited against his CN.
- At the end of each year:
  - The supermarket awards surprise gifts to ten customers who make highest purchase.

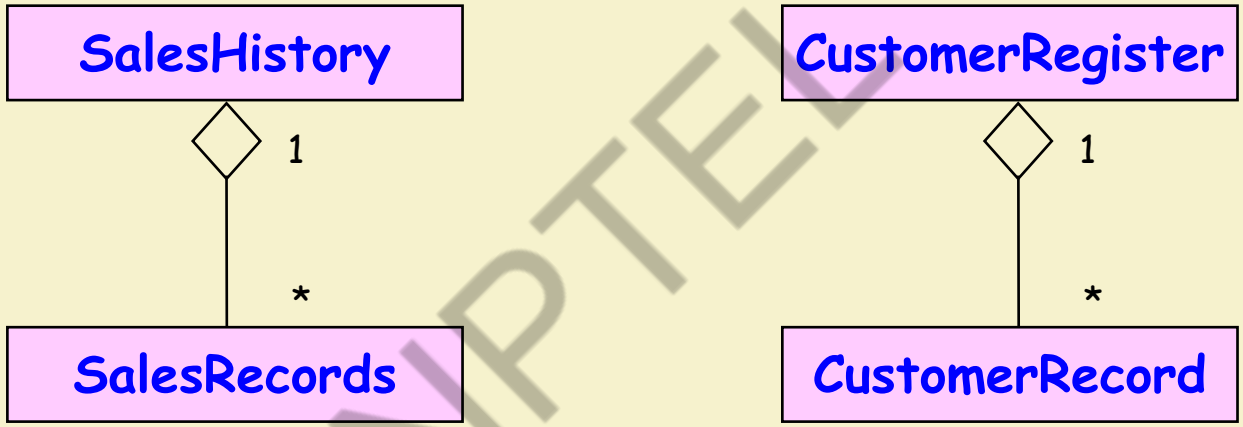
Example 2: Supermarket Prize Scheme

- It also, awards a 22 carat gold coin to every customer:
  - Whose purchases exceed Rs. 10,000.
- The entries against the CN are reset:
  - On the last day of every year after the prize winner's lists are generated.

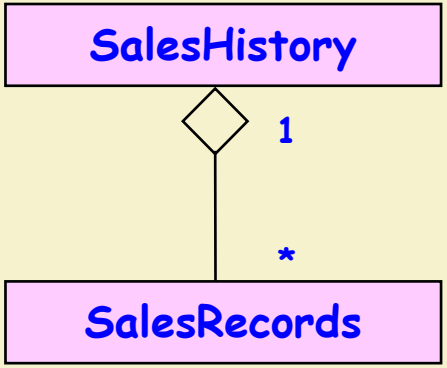


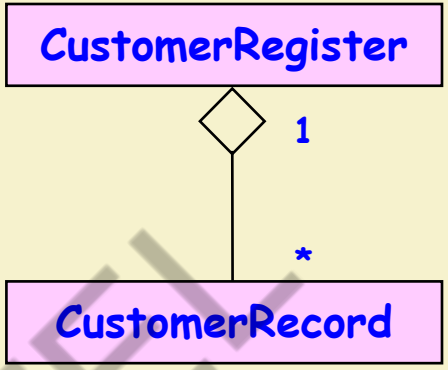
Example 2: Use Case Model

Example 2: Initial Domain Model



Initial domain model





Example 2: Refined Domain Model

RegisterCustomerBoundary

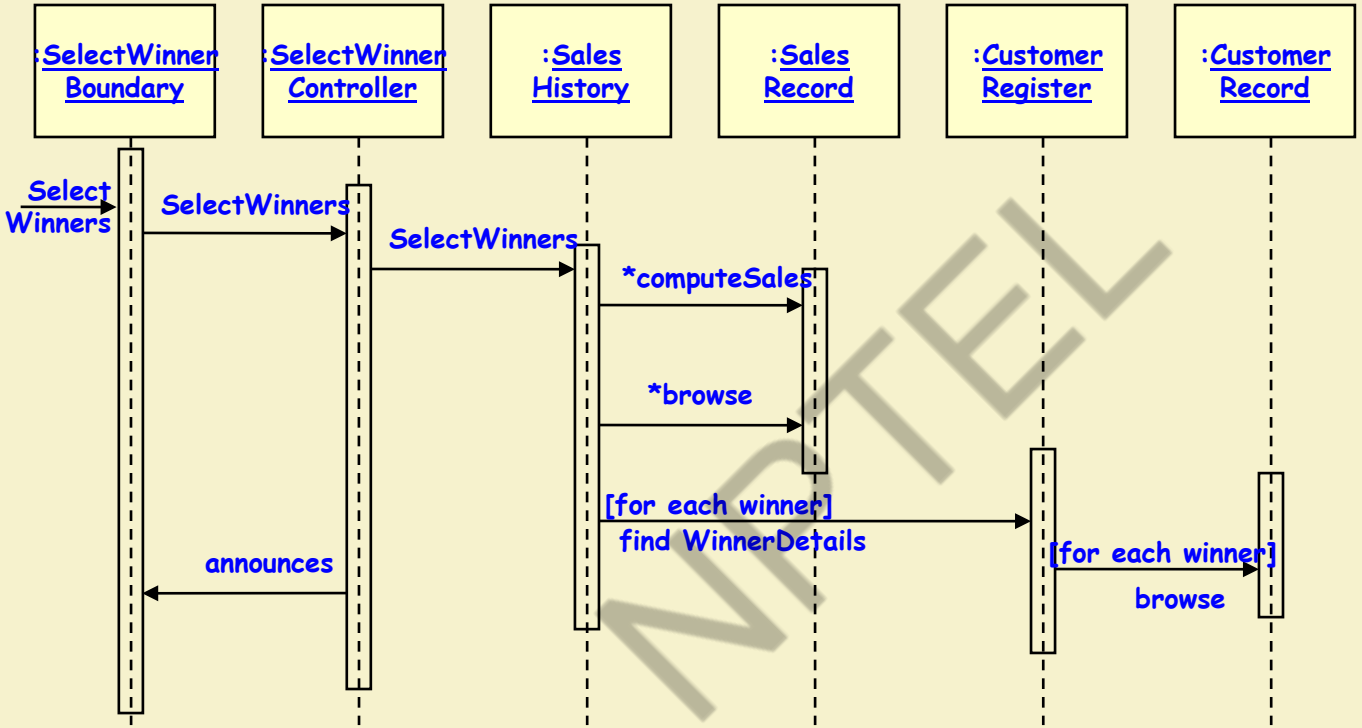
RegisterCustomerController

RegisterSalesBoundary

RegisterSalesController

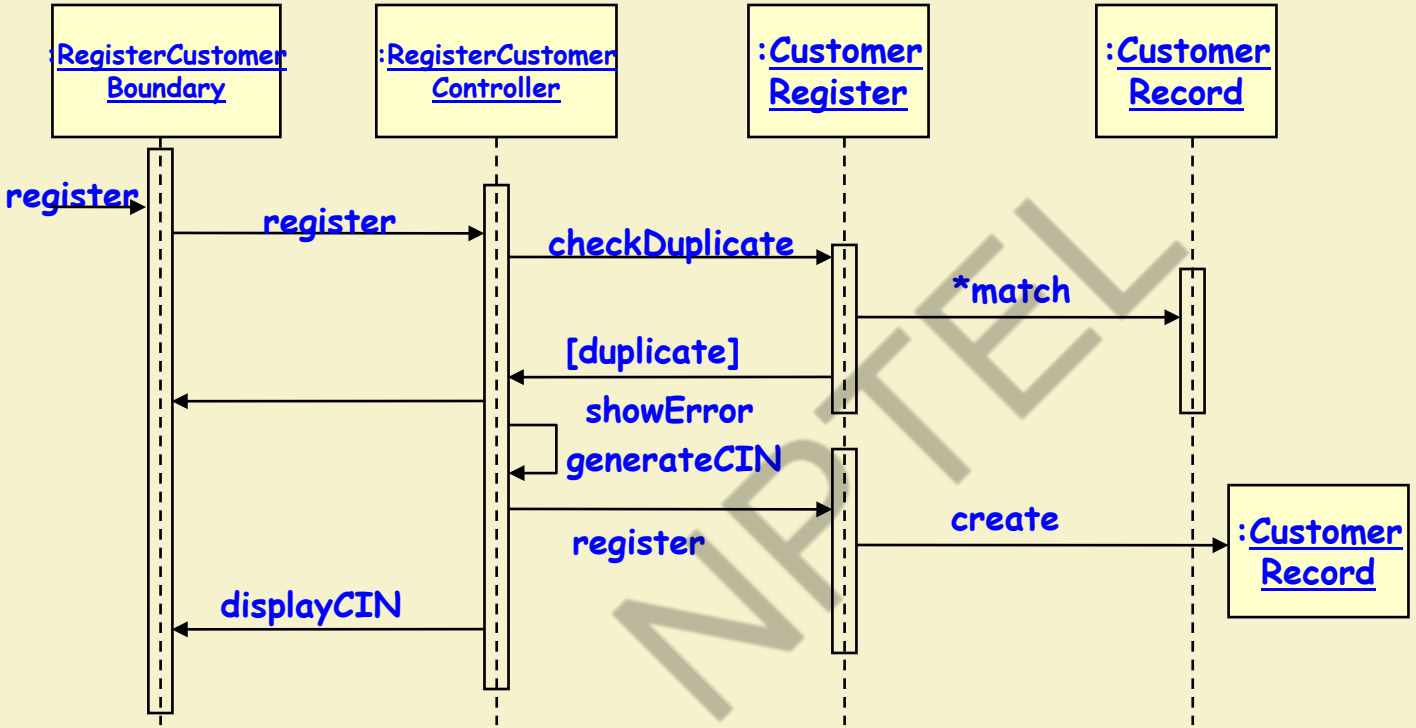
SelectWinnersBoundary

SelectWinnersControllers



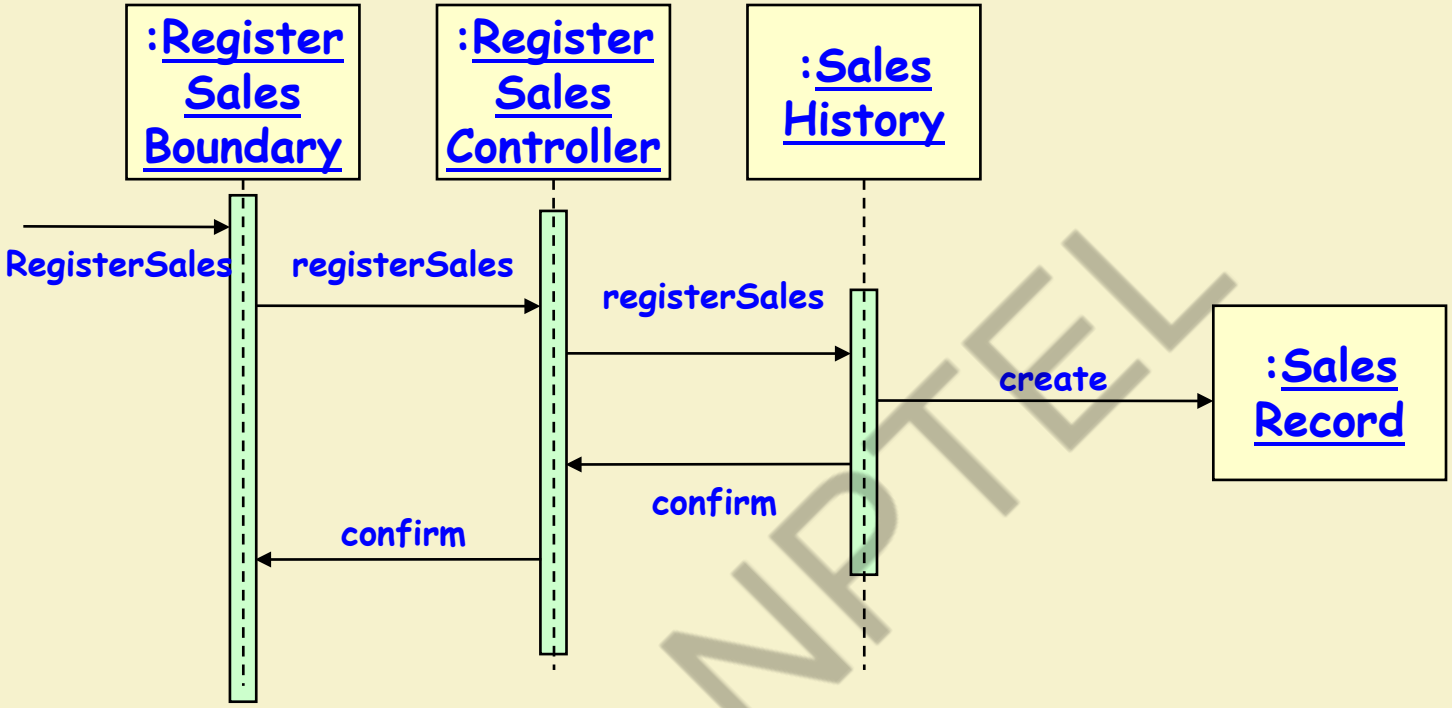
Sequence Diagram for the select winners use case

Example 2: Sequence Diagram for the Select Winners Use Case



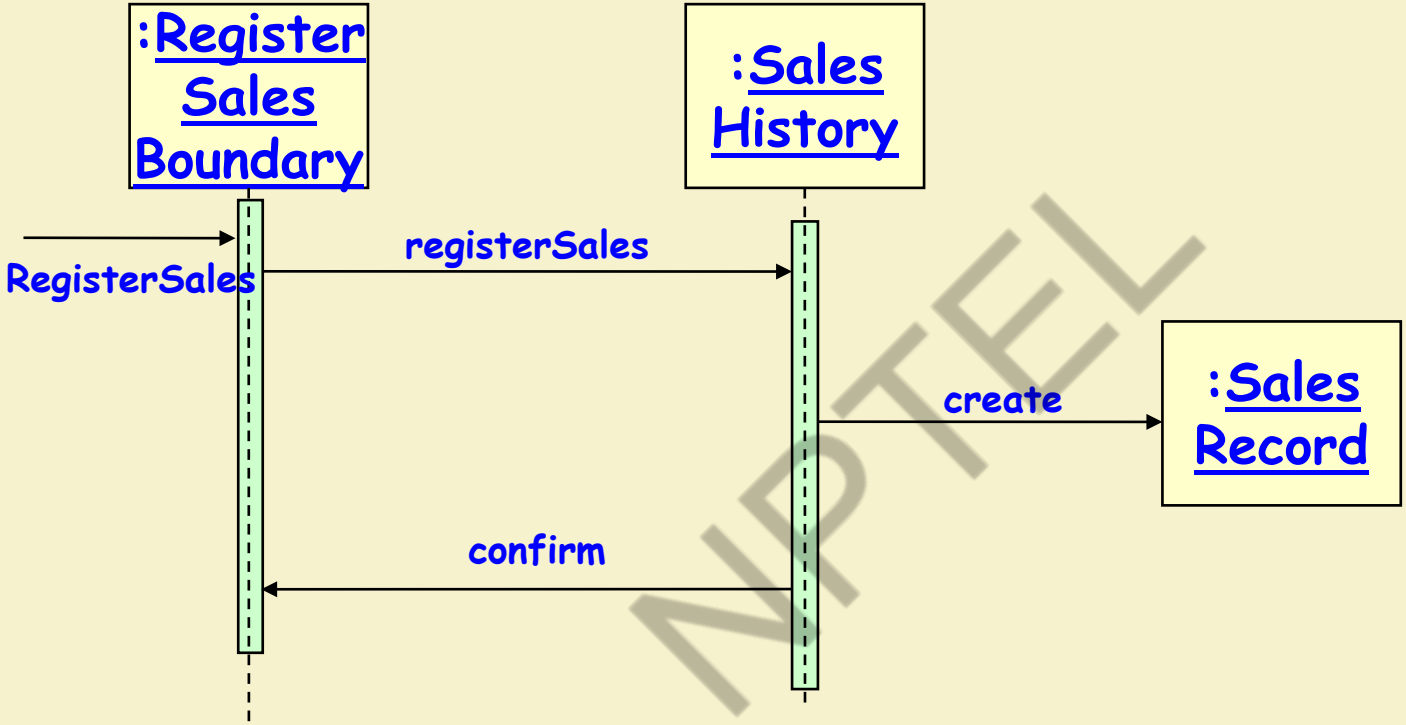
Sequence Diagram for the register customer use case

Example 2: Sequence Diagram for the Register Customer Use Case



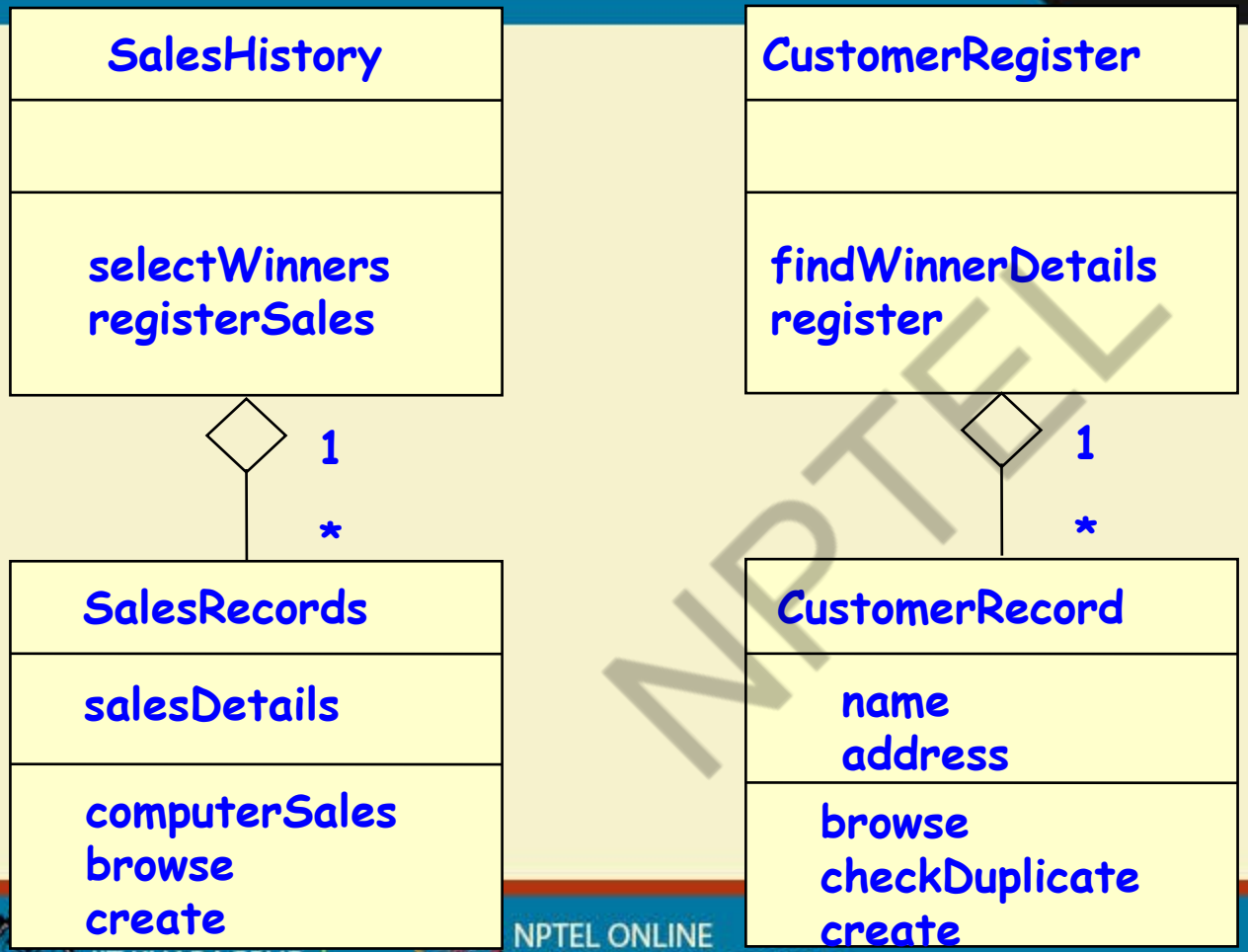
Sequence Diagram for the register sales use case

Example 2: Sequence Diagram for the Register Sales Use Case



Refined Sequence Diagram for the register sales use case

Example 2: Sequence Diagram for the Register Sales Use Case



Example 2: Class Diagram

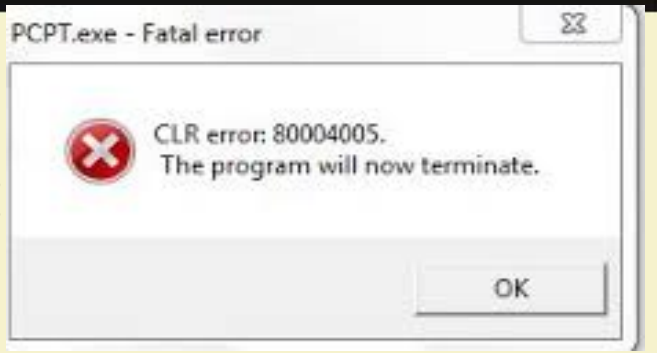
Software Testing

Rajib Mall

CSE Department

IIT KHARAGPUR

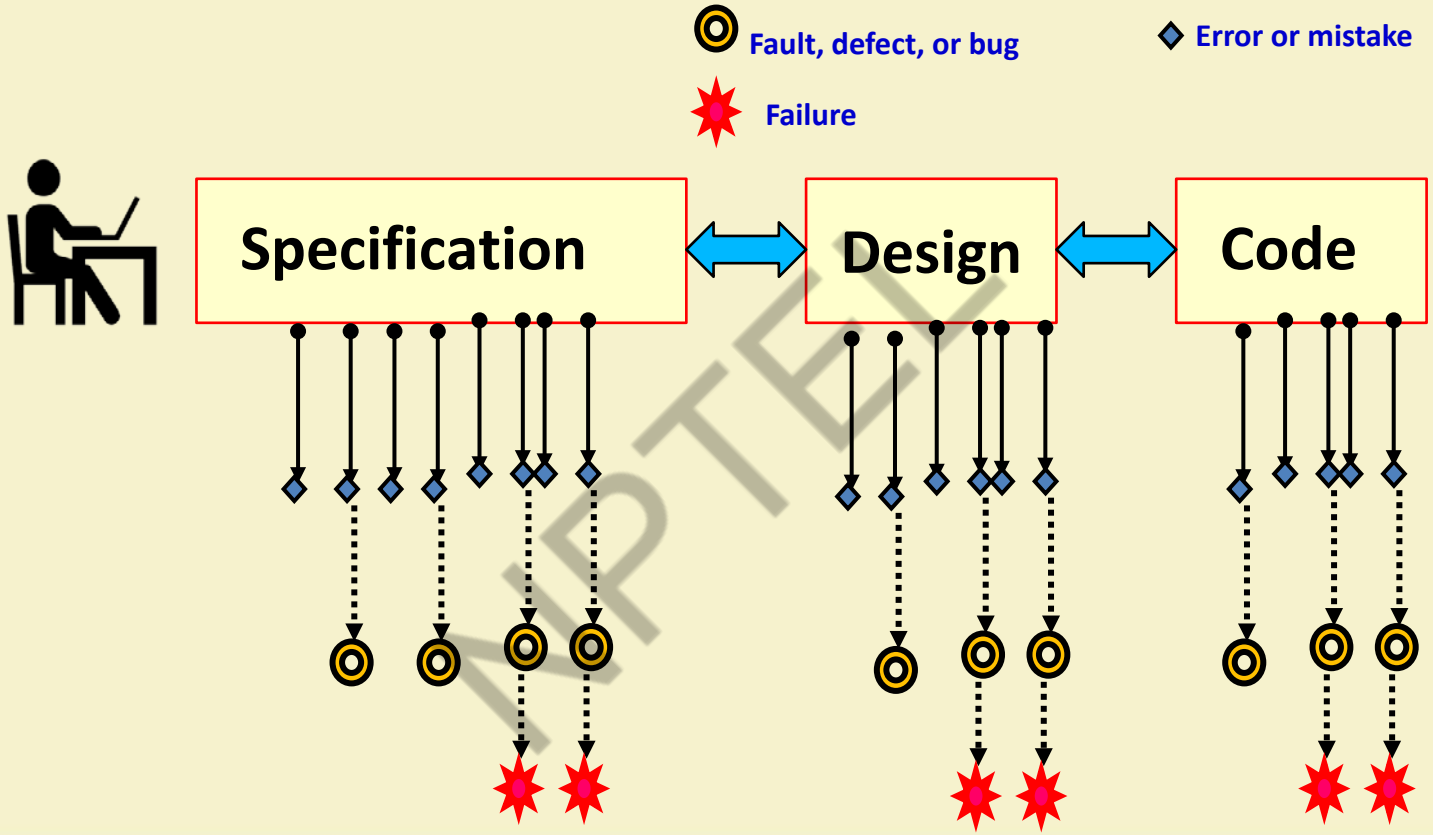
Faults and Failures



- A program may fail during testing:
  - –A manifestation of a fault (also called defect or bug).
  - –**Mere presence of a fault may not lead to a failure.**

Errors, Faults, Failures

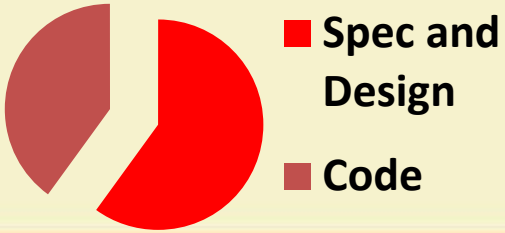
- Programming is human effort-intensive:
  - Therefore, inherently error prone.
- IEEE std 1044, 1993 defined errors and faults as synonyms :
- **IEEE Revision of std 1044 in 2010 introduced finer distinctions:**
  - To support more expressive communications, it distinguished between Errors and Faults

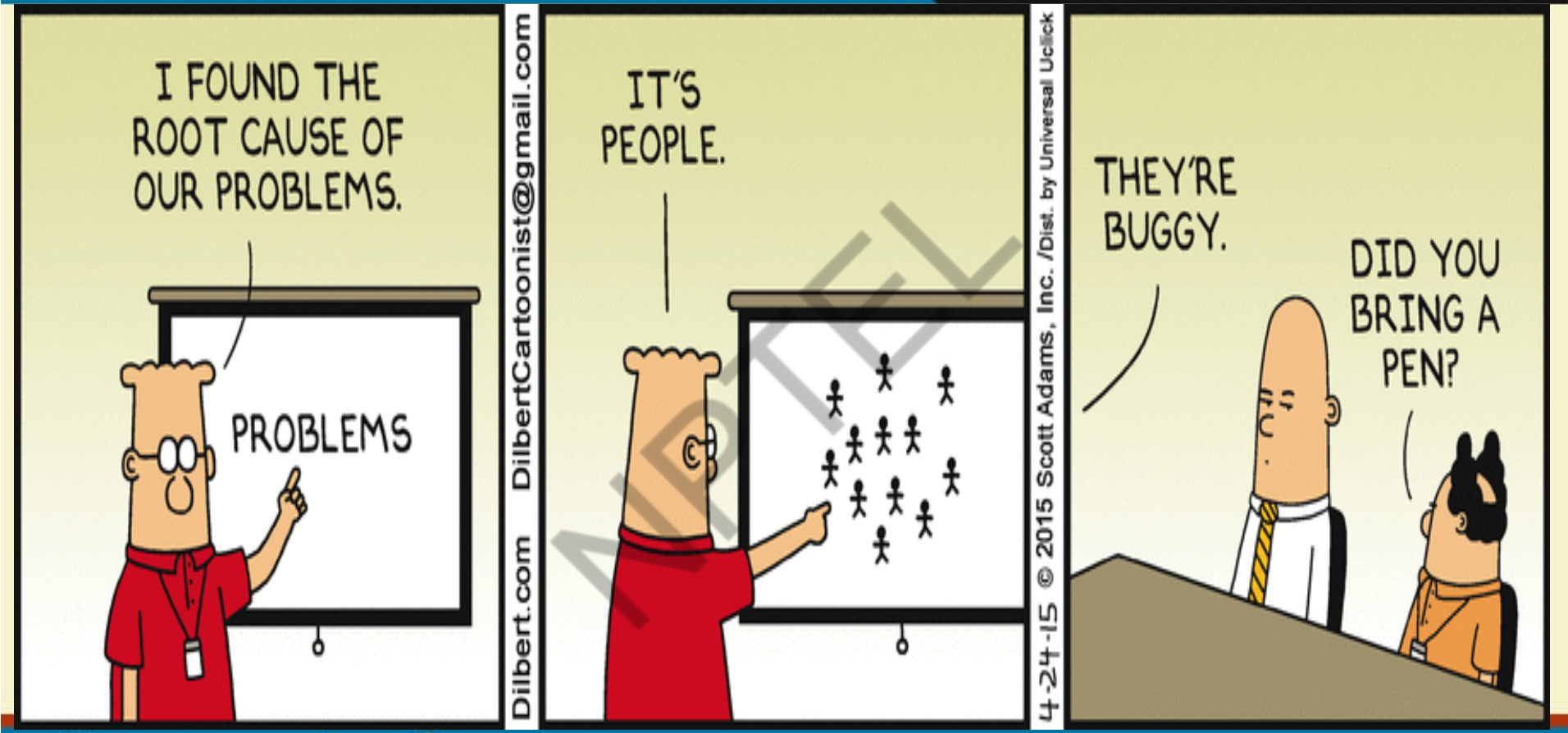

Fault, defect, or bug
Error or mistake
Failure
Specification
Design
Code

A Few Error Facts

- Even experienced programmers make many errors:
  - Avg. 50 bugs per 1000 lines of source code
- Extensively tested software contains:
  - About 1 bug per 1000 lines of source code.
- Bug distribution:
  - 60% spec/design, 40% implementation.

Bug Source



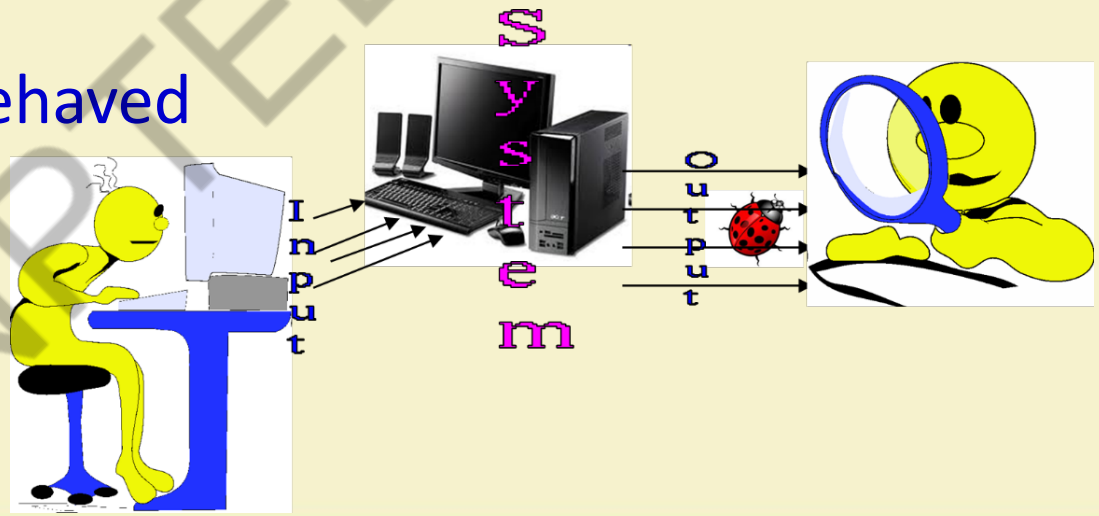

I FOUND THE ROOT CAUSE OF OUR PROBLEMS.
PROBLEMS
Dilbert.com DilbertCartoonist@gmail.com
IT'S PEOPLE.
4-24-15 © 2015 Scott Adams, Inc. /Dist. by Universal Uclick
THEY'RE BUGGY.
DID YOU BRING A PEN?

How to Reduce Bugs?

- Review
- **Testing**
- Formal specification and verification
- Use of development process

How to Test?

- Input test data to the program.
- Observe the output:
  - Check if the program behaved as expected.



Examine Test Result…

- If the program does not behave as expected:
  - –Note the conditions under which it failed (Test report).
  - –Later debug and correct.

Testing Facts

- Consumes the largest effort among all development activities:
  - Largest manpower among all roles
  - Implies more job opportunities
- About 50% development effort
  - But 10% of development time?
  - How?

Testing Facts

- Testing is getting more complex and sophisticated every year.
  - Larger and more complex programs
  - Newer programming paradigms
  - Newer testing techniques
  - Test automation

Testing Perception

- Testing is often viewed as not very challenging --- less preferred by novices, but:
  - Over the years testing has taken a center stage in all types of software development.
  - "**Monkey testing is passe**" --- Large number of innovations have taken place in testing area --- requiring tester to have good knowledge of test techniques.
  - Challenges of test automation

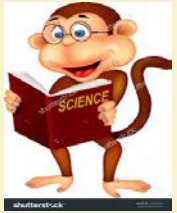
Monkey Testing is Passe…

- Testing through random inputs.
- **Problems:**
  - Many program parts may not get tested.
  - Risky areas of a program may not get tested.
  - The tester may not be able to reproduce the failure.

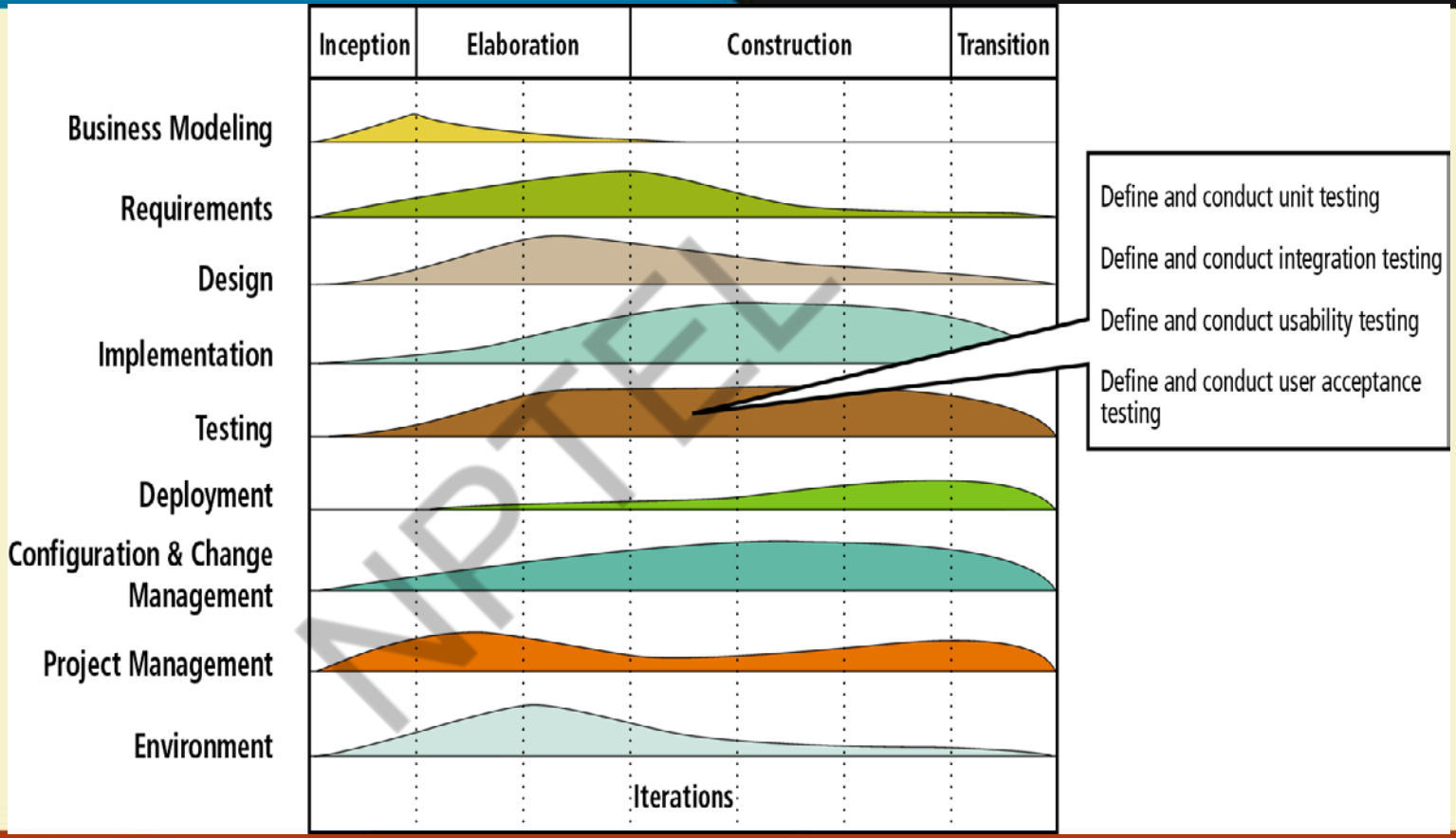
•Two types of monkeys:

•**Dumb monkey**

•**Smart monkey**

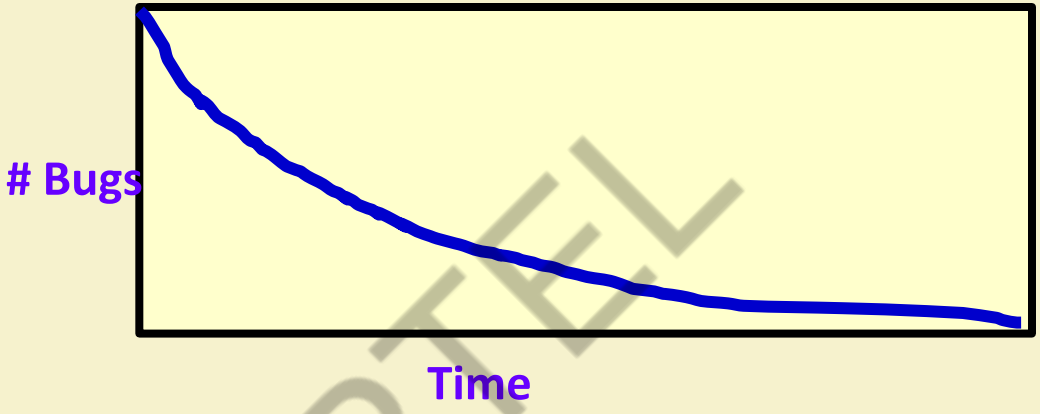


Testing Activities Now Spread Over Entire Life Cycle



Test How Long?

One way:



- Another way:
  - Seed bugs… run test cases
  - See if all (or most) are getting detected

Verification versus Validation

- Verification is the process of determining:
  - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
  - Whether a fully developed system conforms to its SRS document..

Verification versus Validation

- Verification is concerned with phase containment of errors:
  - Whereas, the aim of validation is that the final product is error free.

Verification and Validation Techniques

- Review
- Simulation
- Unit testing
- Integration testing
- System testing

Verification	Validation
Are you building it right?	Have you built the right thing?
Checks whether an artifact conforms to its previous artifact.	Checks the final product against the specification.
Done by developers.	Done by Testers.
Static and dynamic activities: reviews, unit testing.	Dynamic activities: Execute software and check against requirements.

Testing Levels

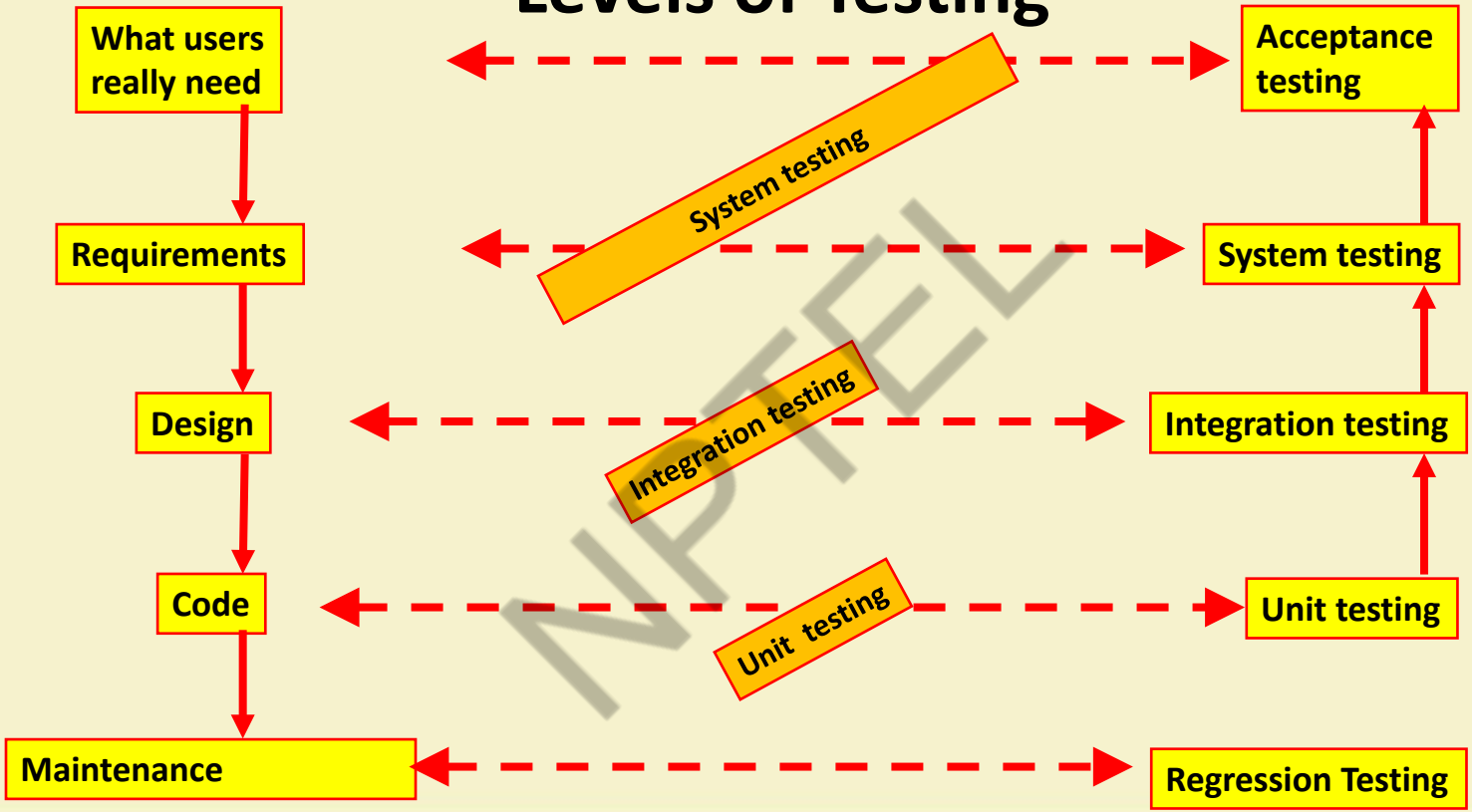
4 Testing Levels

- Software tested at 4 levels:
  - Unit testing
  - Integration testing
  - System testing
  - Regression testing

Test Levels

- **Unit testing**
  - Test each module (unit, or component) independently
  - **Mostly done by developers of the modules**
- **Integration and system testing**
  - Test the system as a whole
  - **Often done by separate testing or QA team**
- **Acceptance testing**
  - **Validation of system functions by the customer**

Levels of Testing



Overview of Activities During System and Integration Testing

- Test Suite Design
- Run test cases
- Check results to detect failures.
- Prepare failure list

Tester

- Debug to locate errors
- Correct errors.

Developer

Quiz 1

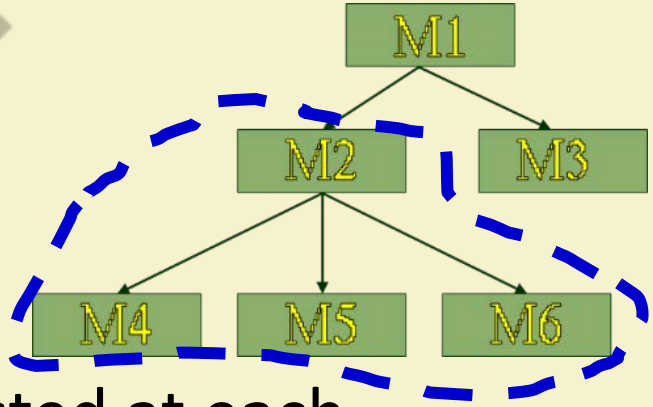
- As testing proceeds more and more bugs are discovered.
  - How to know when to stop testing?
- Give examples of the types of bugs detected during:
  - Unit testing?
  - Integration testing?
  - System testing?

Unit testing

- During unit testing, functions (or modules) are tested in isolation:
  - What if all modules were to be tested together (i.e. system testing)?
    - It would become difficult to determine which module has the error.

Integration Testing

- After modules of a system have been coded and unit tested:
  - –Modules are integrated in steps according to an integration plan
  - –The partially integrated system is tested at each integration step.



Integration and System Testing

- **Integration test evaluates a group of functions or classes:**
  - Identifies interface compatibility, unexpected parameter values or state interactions, and run-time exceptions
  - **System test tests working of the entire system**
- **Smoke test**:
  - System test performed daily or several times a week after every build.

Types of System Testing

- Based on types test:
  - **Functionality test**
  - **Performance test**
- Based on who performs testing:
  - **Alpha**
  - **Beta**
  - **Acceptance test**

Performance test

- Determines whether a system or subsystem meets its non-functional requirements:
  - **Response times**
  - **Throughput**
  - **Usability**
  - **Stress**
  - **Recovery**
  - **Configuration, etc.**

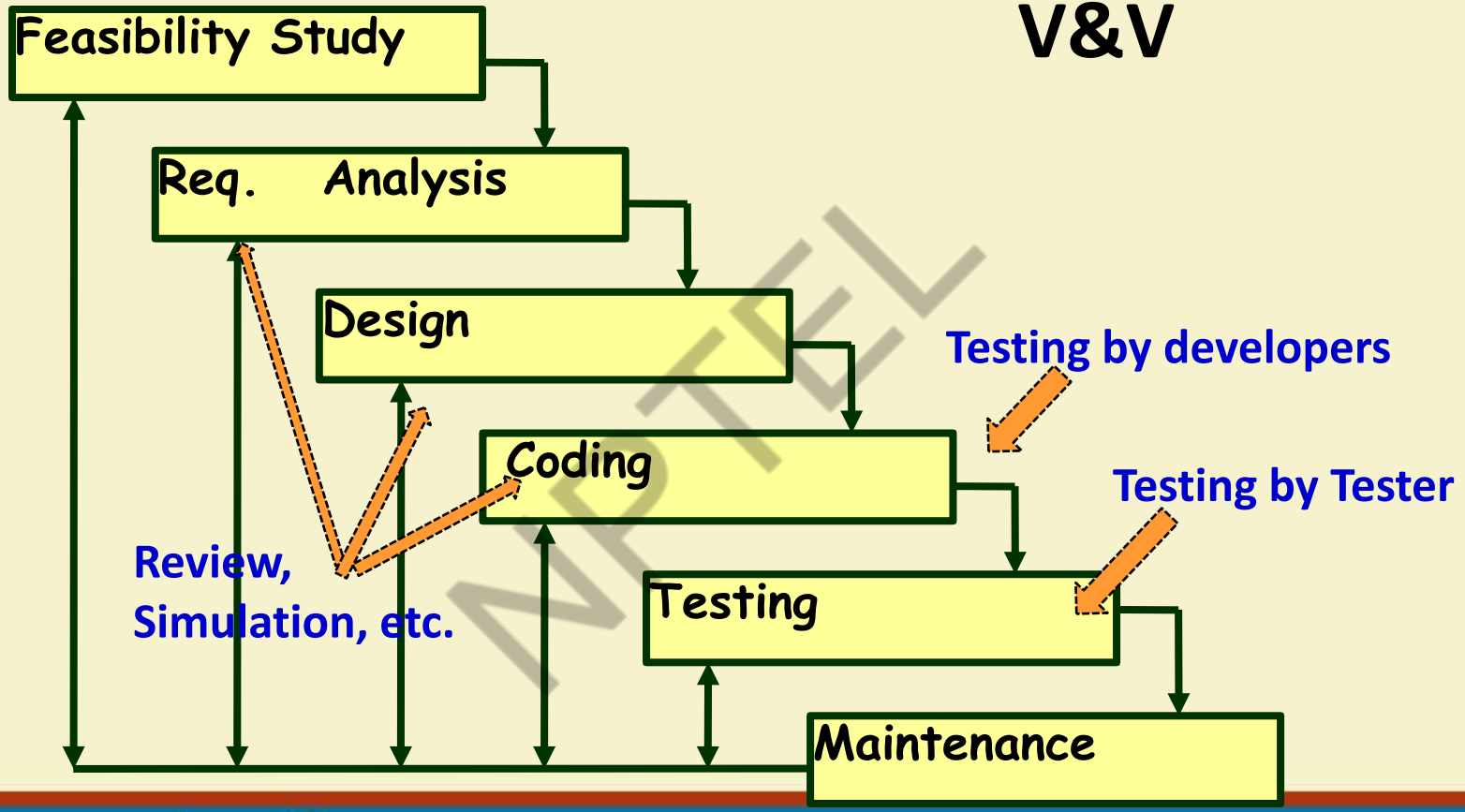
User Acceptance Testing

- User determines whether the system fulfills his requirements
  - **Accepts or rejects delivered system based on the test results.**

Who Tests Software?

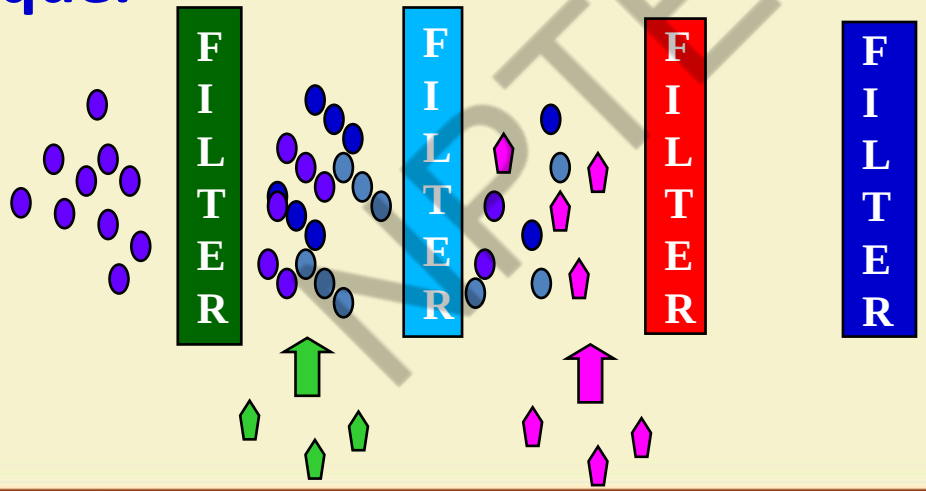
- **Programmers:**
  - Unit testing
  - Test their own or other's programmer's code
- **Users:**
  - Usability and acceptance testing
  - Volunteers are frequently used to test beta versions
- **Test team:**
  - All types of testing except unit and acceptance
  - Develop test plans and strategy

V&V


Feasibility Study
Req. Analysis
Design
Coding
Testing
Maintenance
Testing by developers
Testing by Tester
Review,
Simulation, etc.

Pesticide Effect

- Errors that escape a fault detection technique:
  - **Can not be detected by further applications of that technique.**



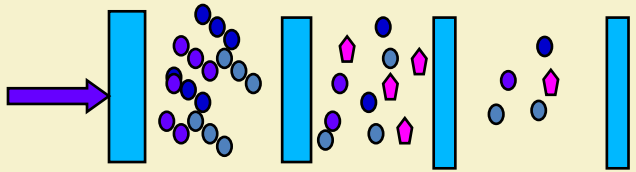
Capers Jones Rule of Thumb

- **Each of software review, inspection, and test step will find 30% of the bugs present.**

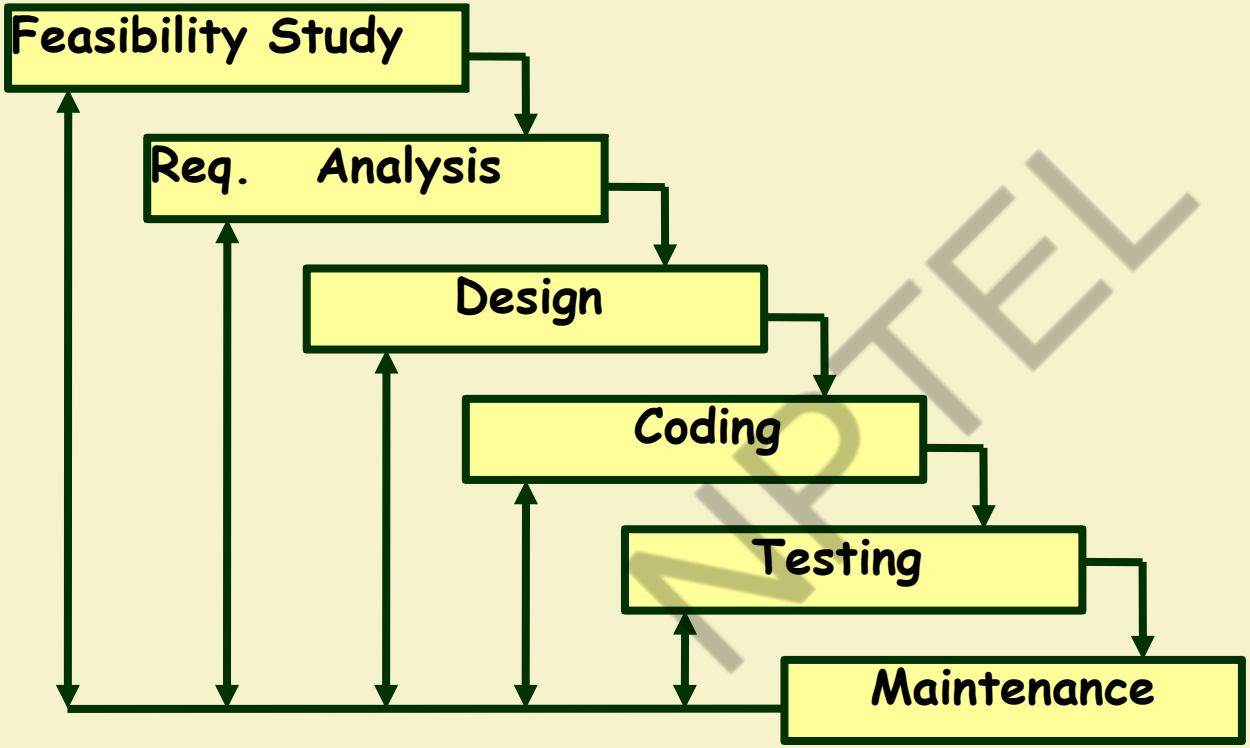
In IEEE Computer, 1996

Pesticide Effect

- Assume to start with 1000 bugs
- We use 4 fault detection techniques :
  - Each detects only 70% bugs existing at that time
  - How many bugs would remain at end?
  - **$1000*(0.3)^4=81$ bugs**



Quiz



1. When are verification undertaken in waterfall model?
2. When is testing undertaken in waterfall model?
3. When is validation undertaken in waterfall model?

Basic Concepts in Testing

- Several independent studies [Jones],[schroeder], etc. conclude:
  - **85% errors get removed at the end of a typical testing process.**
  - **Why not more?**
  - **All practical test techniques are basically heuristics… they help to reduce bugs… but do not guarantee complete bug removal…**

How Many Latent Errors?

Test Cases

- Each test case typically tries to establish correct working of some functionality:
  - Executes (covers) some program elements.
  - For certain restricted types of faults, fault-based testing can be used.

Test data versus test cases

- **Test data:**
  - Inputs used to test the system
- **Test cases:**
  - Inputs to test the system,
  - State of the software, and
  - The predicted outputs from the inputs

Test Cases and Test Suites

- A test case is a triplet [I,S,O]
  - I is the data to be input to the system,
  - S is the state of the system at which the data will be input,
  - O is the expected output of the system.

Test Cases and Test Suites

- Test a software using a set of carefully designed test cases:
  - The set of all test cases is called the **test suite.**

What are Negative Test Cases?

- **Purpose:**
  - Helps to ensure that the application gracefully handles invalid and unexpected user inputs and the application does not crash.
- **Example:**
  - If user types letter in a numeric field, it should not crash but politely display the message: **"incorrect data type, please enter a number..."**

Test Execution Example: Return Book

- **Test case [I,S,O]**

1. **Set the program in the required state:** Book record created, member record created, Book issued
2. **Give the defined input:** Select renew book option and request renew for a further 2 week period.
3. **Observe the output:**
   - Compare it to the expected output.

Sample: Recording of Test Case & Results

Test Case number

Test Case author

Test purpose

Pre-condition:

Test inputs:

Expected outputs (if any):

Post-condition:

Test Execution history:

- Test execution date
- Person executing Test
- Test execution result (s) : Pass/Fail
  - If failed : Failure information and fix status

Test Team- Human Resources

- **Test Planning:** Experienced people
- **Test scenario and test case design:** Experienced and test qualified people
- **Test execution:** semi-experienced to inexperienced
- **Test result analysis:** experienced people
- **Test tool support:** experienced people
- May include external people:
  - **Users**
  - **Industry experts**

Why Design of Test Cases?

- Exhaustive testing of any non-trivial system is impractical:
  - –Input data domain is extremely large.
- Design an optimal test suite, meaning:
  - –Of reasonable size, and
  - –Uncovers as many errors as possible.

Design of Test Cases

- If test cases are selected randomly:
  - Many test cases would not contribute to the significance of the test suite,
  - Would only detect errors that are already detected by other test cases in the suite.
- Therefore, the number of test cases in a randomly selected test suite:
  - Does not indicate the effectiveness of testing.

Design of Test Cases

- Testing a system using a large number of randomly selected test cases:
  - **Does not mean that most errors in the system will be uncovered.**
- Consider following example:
  - Find the maximum of two integers x and y.

Design of Test Cases

- The code has a simple programming error:
- **If (x>y) max = x;**
  else max = x; // should be max=y;
- Test suite {(x=3,y=2);(x=2,y=3)} can detect the bug,
- A larger test suite {(x=3,y=2);(x=4,y=3); (x=5,y=1)} does not detect the bug.

- Before testing activities start, a test plan is developed.
- The test plan documents the following:
  - Features to be tested
  - Features not to be tested
  - Test strategy
  - Test suspension criteria
  - stopping criteria
  - Test effort
  - Test schedule

Test Plan