SENG 301 Software Analysis and Design

Midterm Review - May Mahmoud, Lakshya Tandon and Sydney Pratte

Modeling

May Mahmoud

Modeling

- Why use Models?
  - “Just enough modeling is the target to strive for, not more, not less”

- Good vs Bad Models

Type of Models

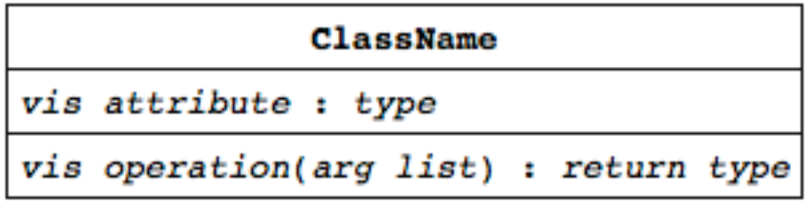
- Structural Model
  - Class Diagram
- Behavioral Model
  - Sequence Diagram
  - State Diagram

Structural Diagram

- Used to represent
  - Classes (attributes and operations)
  - Relationship between types
  - Object
  - Packages

- Not used to represent run-time behavior

Class Diagram

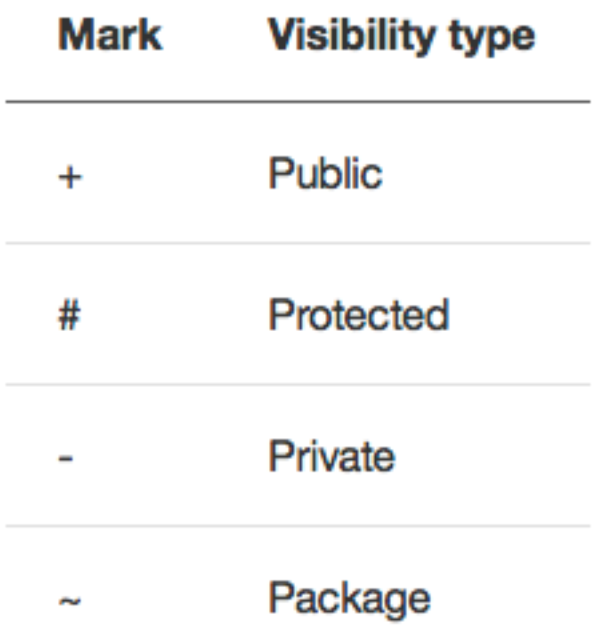


ClassName

vis attribute : type

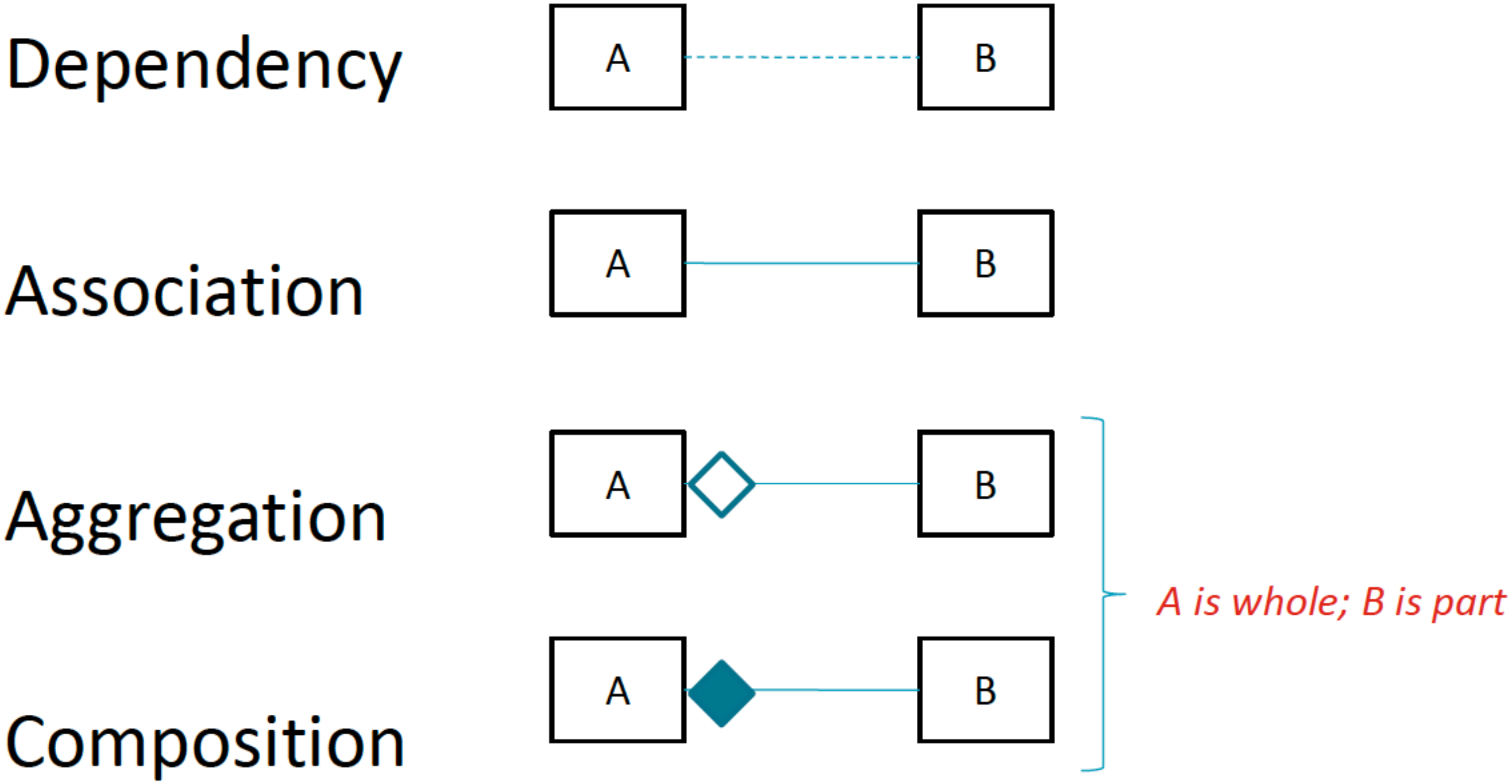
vis operation(arg list) : return type

Visibility – Class Diagram



Mark	Visibility type
+	Public
#	Protected
-	Private
~	Package

Relationships



Dependency

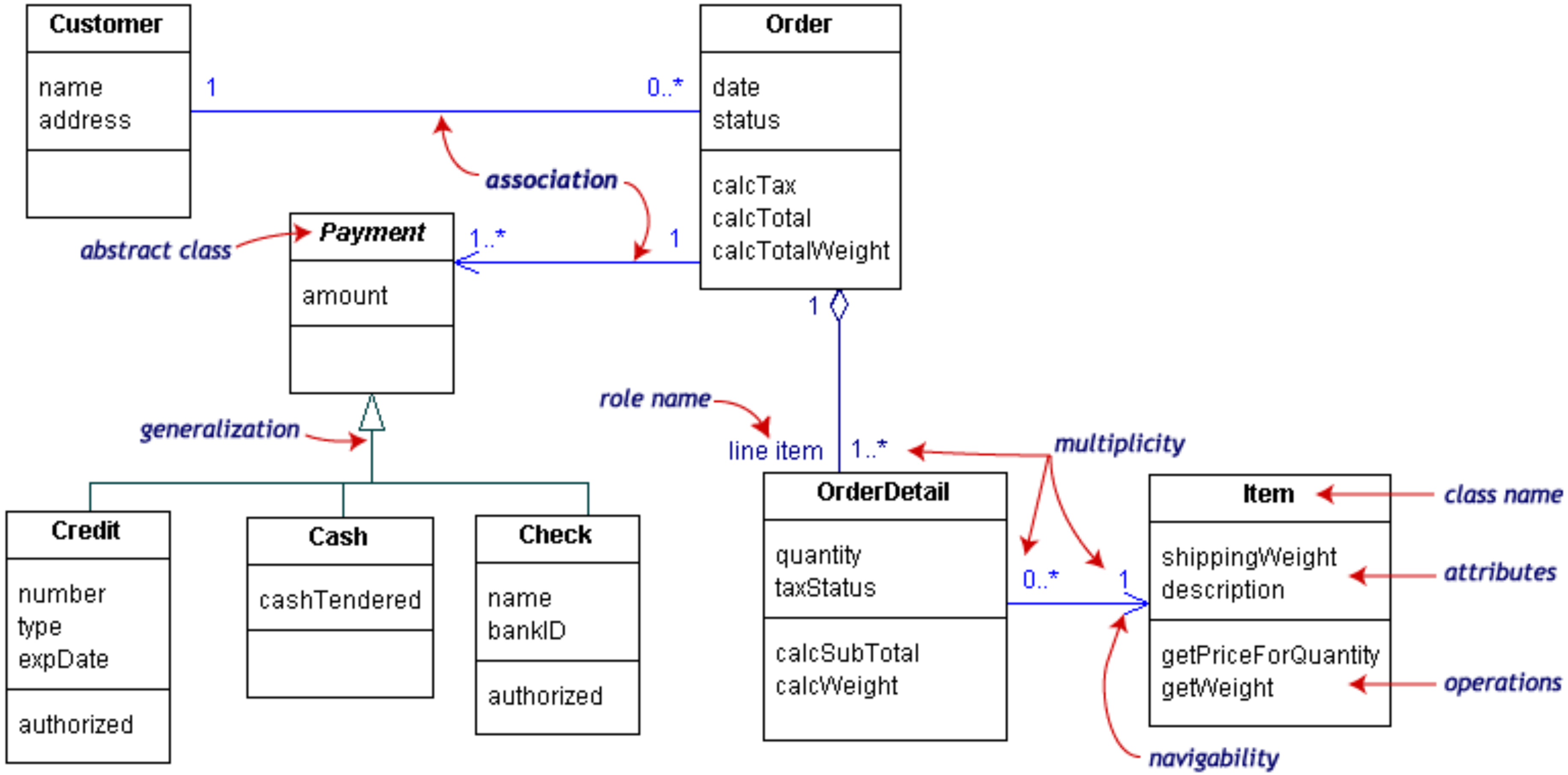
Association

Aggregation

Composition

A is whole; B is part

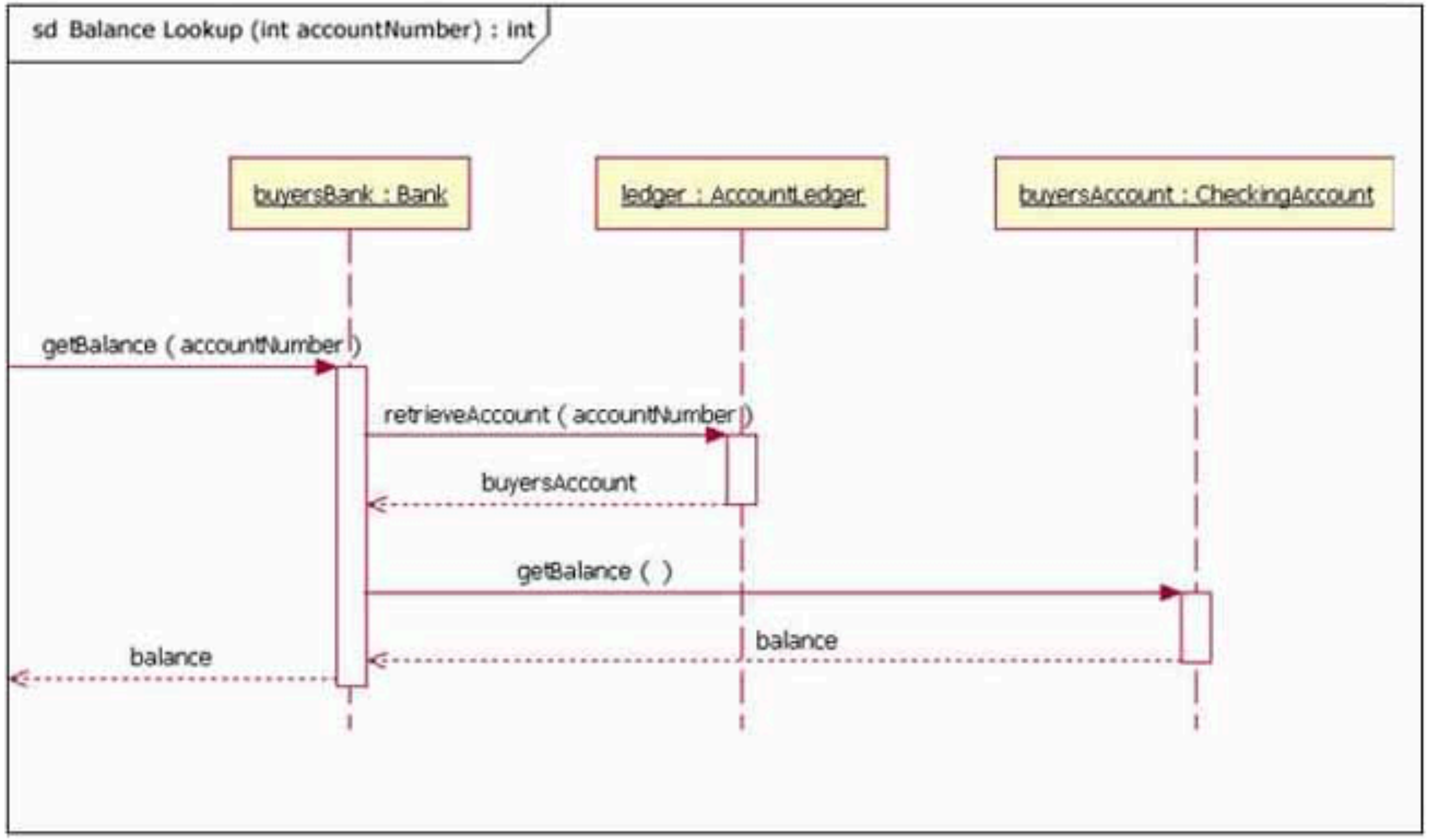
Class Diagram - Sample



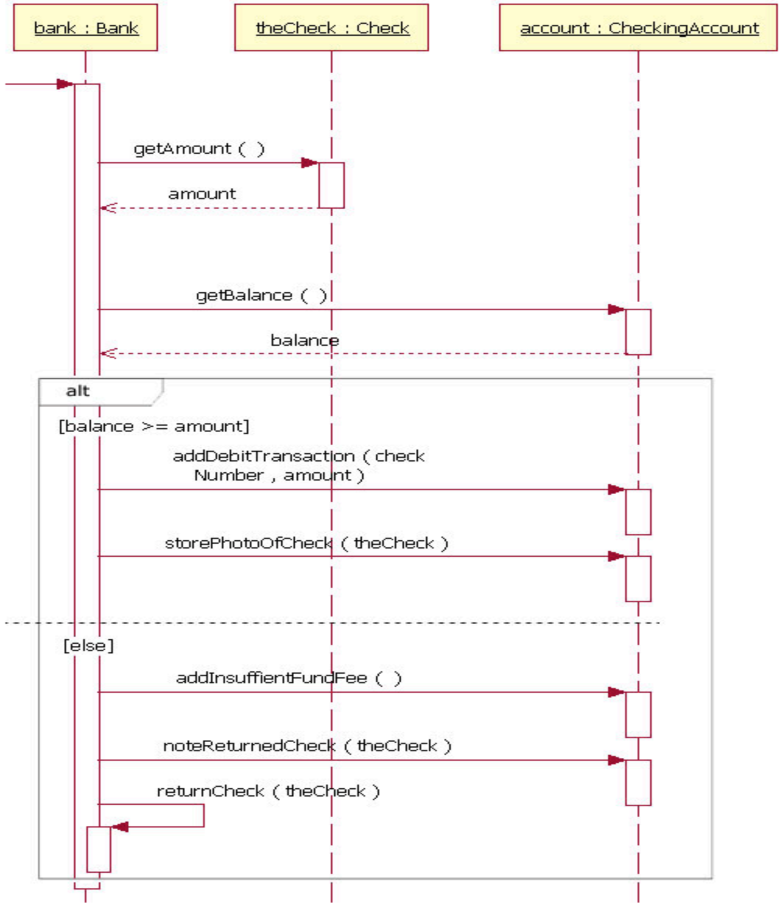
Behavioral Models

- Model the dynamic aspects of a system
- Sequence Diagram
- State Machine Diagram

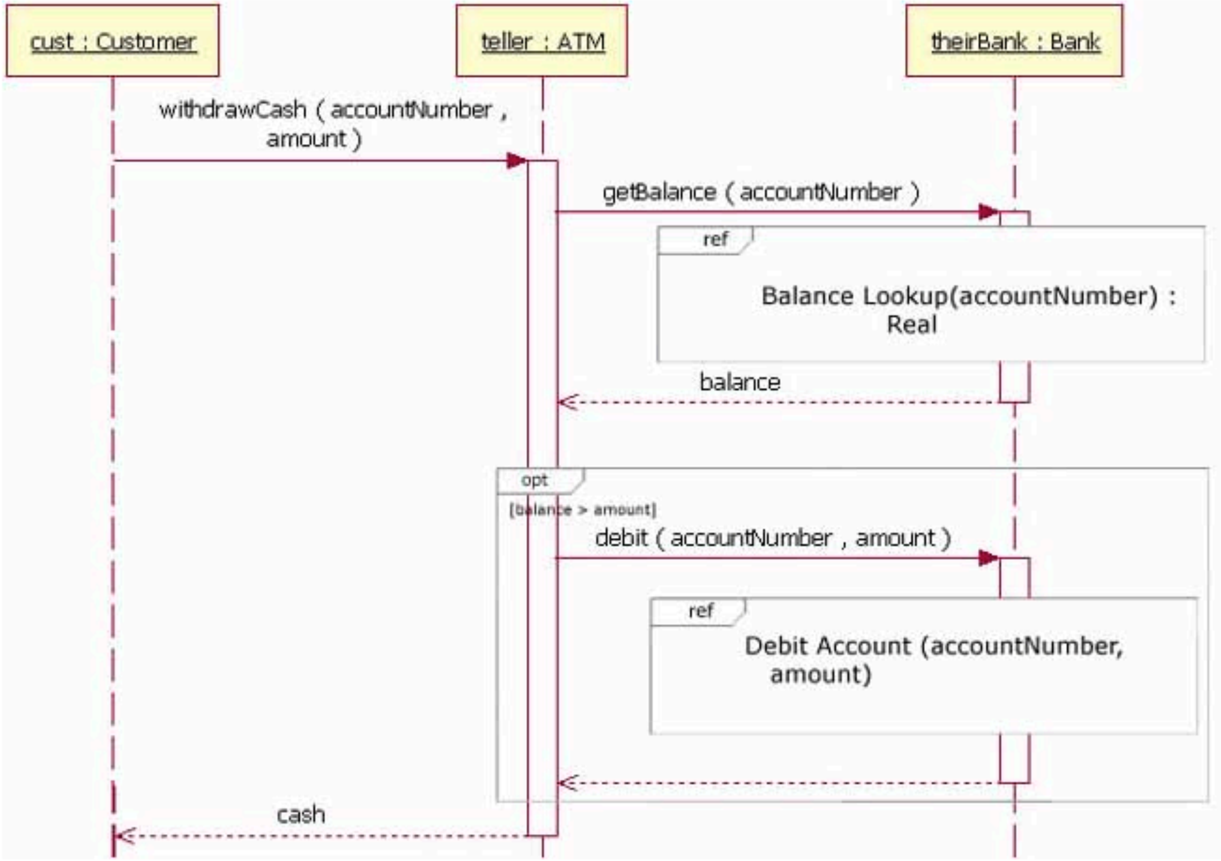
Sequence Diagram


sd Balance Lookup (int accountNumber) : int
buyersBank : Bank
ledger : AccountLedger
buyersAccount : CheckingAccount
getBalance (accountNumber)
retrieveAccount (accountNumber)
buyersAccount
getBalance ( )
balance
balance

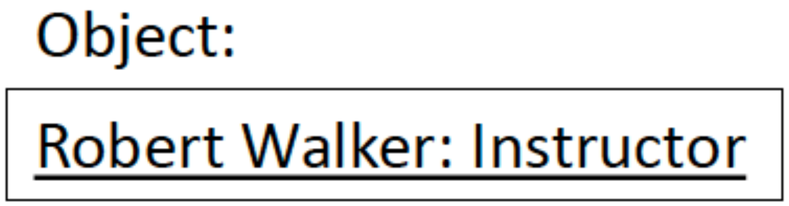
Sequence Diagram - Alternatives



Sequence Diagram- Referencing another sequence diagram

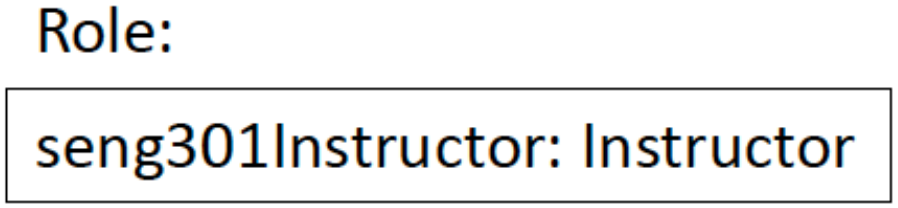

cust : Customer
teller : ATM
theirBank : Bank
withdrawCash (accountNumber , amount)
getBalance (accountNumber)
ref
Balance Lookup(accountNumber) : Real
balance
opt
[balance > amount]
debit (accountNumber , amount)
ref
Debit Account (accountNumber, amount)
cash

Sequence Diagram- Objects vs Roles



Object:

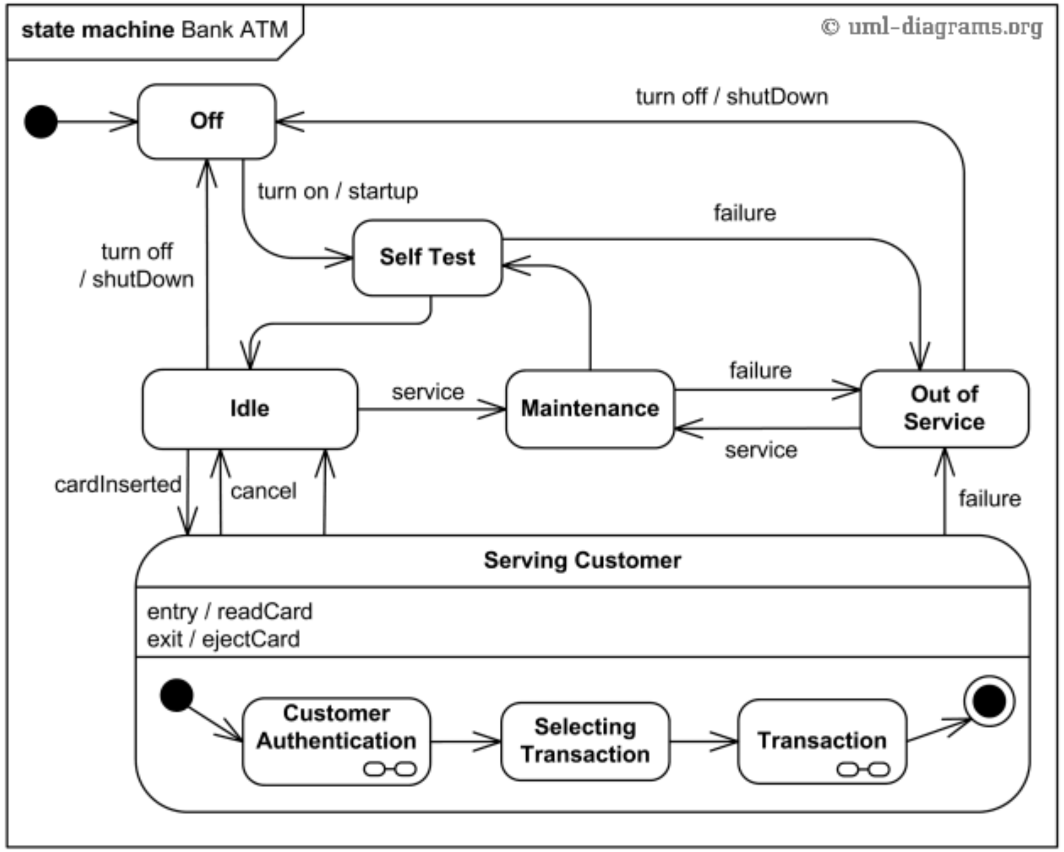
Robert Walker: Instructor



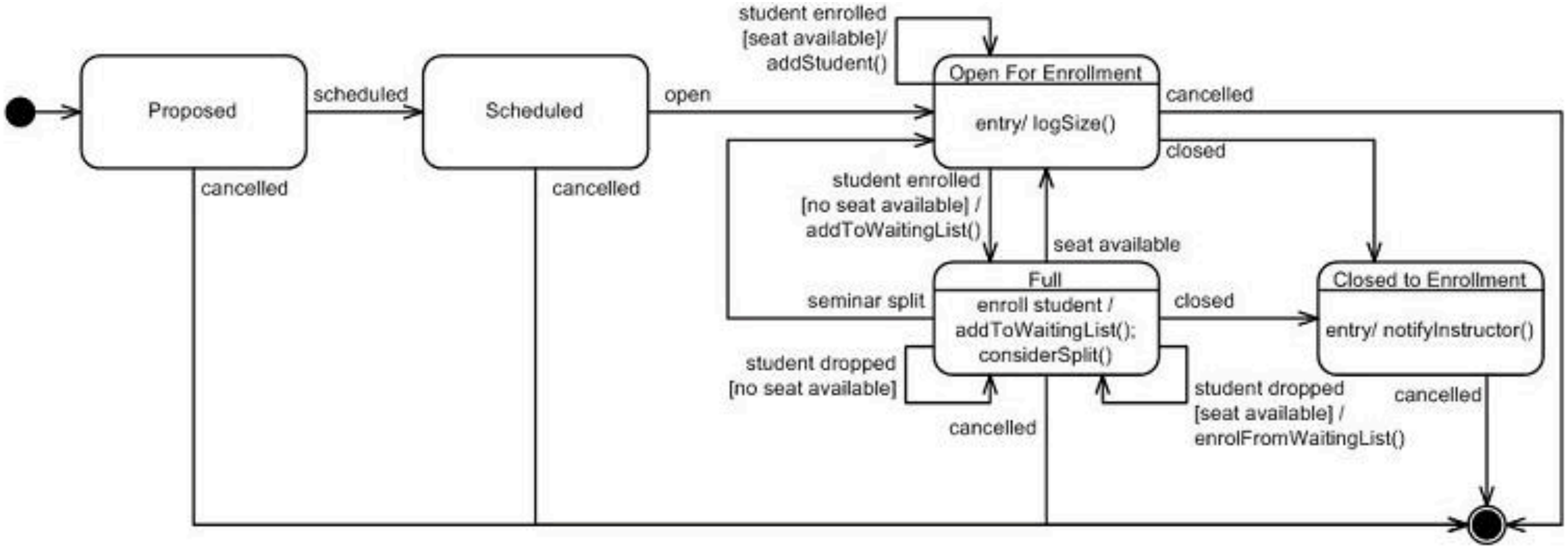
Role:

seng301Instructor: Instructor

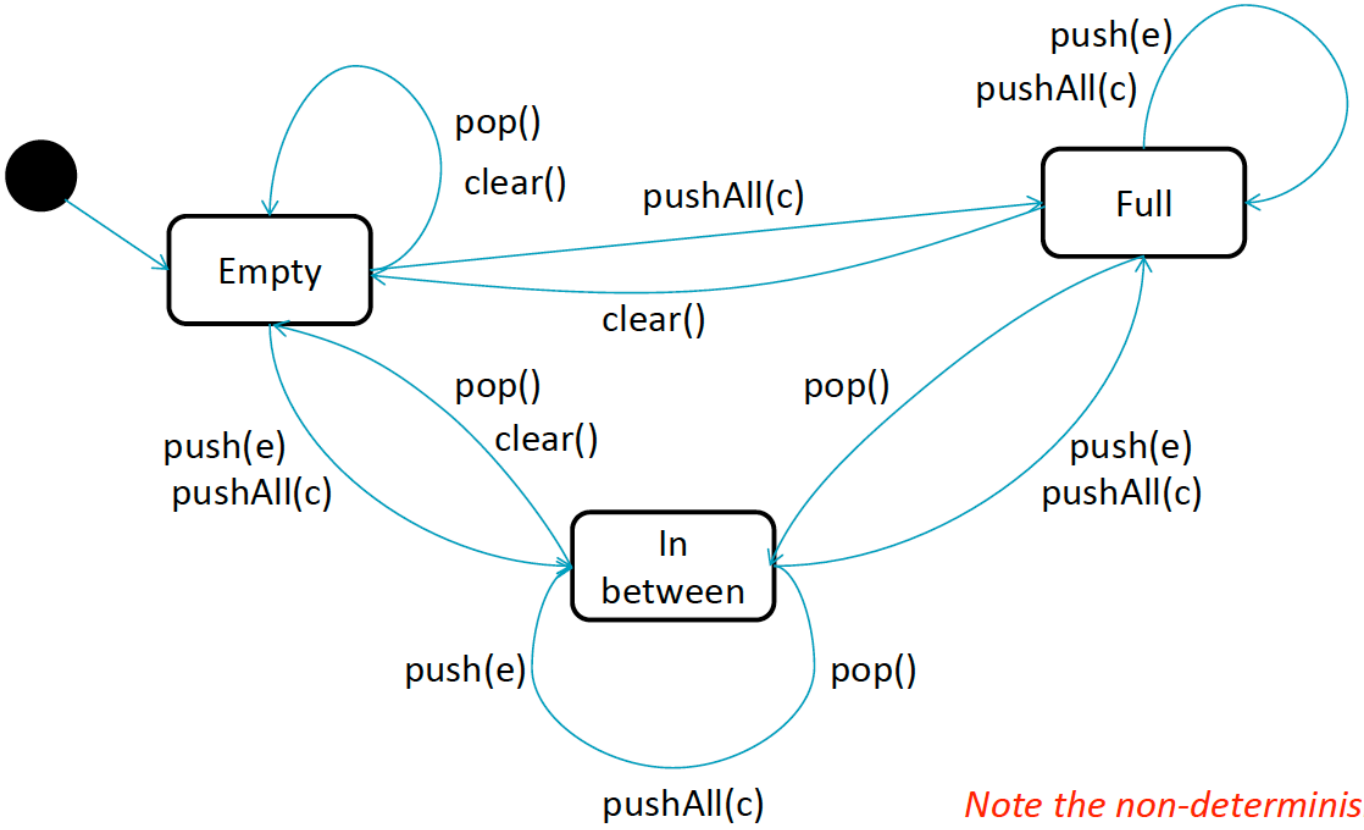
State Machine Diagram



State Machine Diagram


Proposed
scheduled
Scheduled
open
cancelled
cancelled
student enrolled [seat available]/ addStudent()
Open For Enrollment
entry/ logSize()
cancelled
closed
student enrolled [no seat available] / addToWaitingList()
seat available
Full
enroll student / addToWaitingList(); considerSplit()
seminar split
closed
Closed to Enrollment
entry/ notifyInstructor()
student dropped [no seat available]
cancelled
student dropped [seat available] / enrolFromWaitingList()
cancelled

Non-determinism



Note the non-determinism.

Requirements

Sydney Pratte

The Basics

- WHAT the system does
  - Not how it should be done
- System requirements outline what the system should do and the end goal.

Functional vs. Non-Functional

- Functional requirements
  - Features of the system

- Non-functional requirements
  - Performance
  - Reliability
  - Legal issues
  - Quality
  - Usability
  - Maintainability
  - Security
  - …

Requirements Representations

- Natural language
- Structured natural language
- User stories
- Use cases
- Formal specification

Natural Language Requirements

- Statements in natural language of what the system should and shouldn’t do
- Easy to read

Structured Natural Language Requirements

- Similar to Natural language requirements
- More structured

Scenarios (user stories, use cases)

- Describes a specific interaction a user can have with the system
- Need multiple scenarios to cover all possibilities

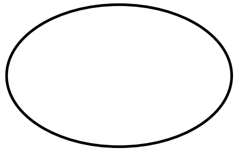
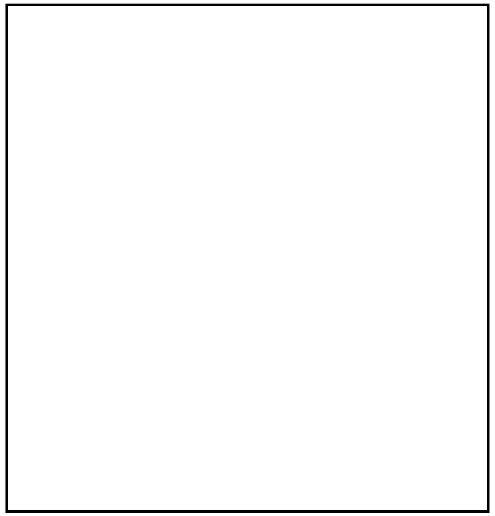
User Stories

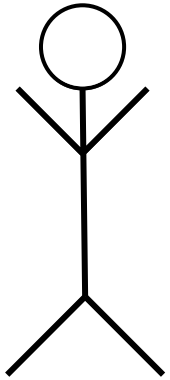
- Describes what the user wants
  - Written by the user
  - Structured as:

  As a <role>, I want <desired> so that <benefit>
- Epic user stories are user stories that can be broken down to smaller, more useful stories.

Use Cases

- Usually represents a major piece of functionality
- Made up of scenarios
- Uses UML symbols

symbol for use cases  symbol for system 

symbol for actor 

Use Case Description

Name : name of the use case described

Participating actor(s): who is interacting with the system

Precondition: the state of the system before

Main flow of events:

1. The steps the actor and system take in the use case

Postcondition: state of the system after the use case

Alternative flow: another path that can be taken during the use case

Postcondition: state of the system after the alternate flow

Requirements Analysis

- Correctness
- Completeness
- Consistency
- Reality
- Verifiability
- Cost
- Priority

Example – ATM Machine

Name: Enter pin

Actor: Bank Customer

Precondition: Personal bank card has been inserted into the machine

Main Flow of Events:

1. Customer enters 4 digit pin
2. Customer presses enter
3. System validates pin

Postcondition: Customer has access to their account

Alternative flow: Wrong pin

1. Customer enters the wrong pin
2. Customer presses enter
3. System validation fails
4. System re-prompts for pin

Postcondition: System prompts for pin again

Example – ATM Machine

As a bank customer, I want to access my account on an ATM, so that I can withdraw funds.

Example – ATM Machine

Customers can withdraw and deposit money into their personal accounts. Customer has to insert their personal banking card. Customer must enter a valid 4-digit pin to access their account. Customers can only withdraw money if the funds are available.

Example – ATM Machine

- Validation:
  - ATM accepts valid bank cards
  - ATM accepts a four digit pin and
- Withdrawal:
  - Customers can select a desired amount to withdraw from their account
  - If the ATM contains the desired funds and the funds exist in the customers account the money is delivered to the customer
  - …

Testing

Lakshya Tandon

Black Box Testing

- It treats software under test as a black box without knowing its internals.
- Tester is aware of what the program should do but does not have the knowledge of how it does it.
- Black-box testing is most commonly used type of testing in traditional organizations that have testers as a separate department, especially when they are not proficient in coding and have difficulties to understand the code.
- Also known as functional testing.
- It provides **external perspective** of the software under test.

Advantages

- Efficient for large segments of code
- Code access is not required
- Separation between user’s and developer’s perspectives

Disadvantages

- Limited coverage since only a fraction of test scenarios is performed.
- Inefficient testing due to tester’s luck of knowledge about software internals.
- Blind coverage since tester has limited knowledge about the application

White Box testing

- White box testing looks inside the software that is being tested and uses that knowledge as part of the testing process.
- For example, exception is thrown under certain conditions, test might want to reproduce those conditions.
- Requires internal knowledge of the system and programming skills.
- Also known as clear box testing, glass box testing, transparent box testing, and structural testing

Advantages

- Efficient in finding errors and problems
- Allows finding hidden errors
- Helps optimizing the code
- Due to required internal knowledge of the software, maximum coverage is obtained

Disadvantages

- Might not find unimplemented or missing features
- Requires high level knowledge of internals of the software under test
- Requires code access

Path Testing

- Path Testing is a structural testing method based on the source code or algorithm and NOT based on the specifications.

Path Testing Techniques

- **Control Flow Graph (CFG) -** The Program is converted into Flow graphs by representing the code into nodes and edges.

- **Decision to Decision path (D-D) -** The CFG can be broken into various Decision to Decision paths and then collapsed into individual nodes.

- **Independent (basis) paths -** Independent path is a path through a DD-path graph which cannot be reproduced from other paths by other methods.

Boundary Testing

- Boundary value analysis is a type of black box or specification based testing technique in which tests are performed using the boundary values.

Lets take an example – Boundary Testing

- To pass an exam a student needs 50%, for merit he needs 75% and for distinction he needs 90%

- The Boundary values would be:
  - 49 and 50 for pass
  - 74 and 75 for merit
  - 89 and 90 distinction

Equivalence class testing

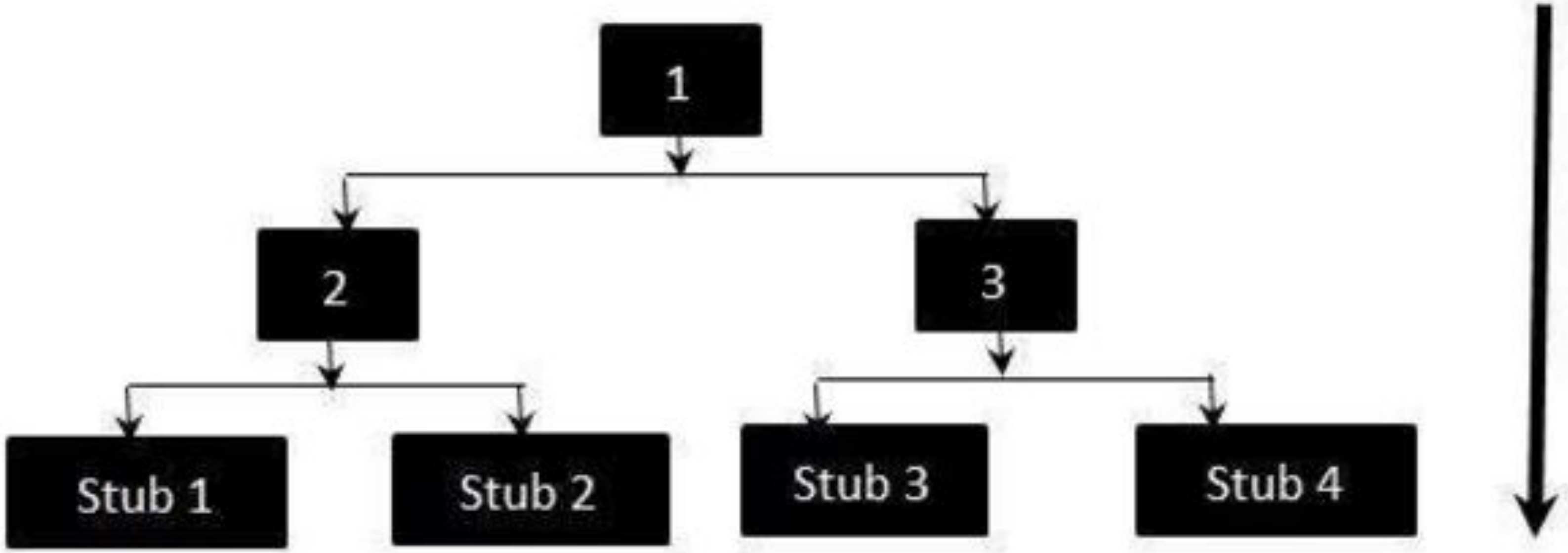
- A technique that divides the input test data of the application under test into each partition at least once of equivalent data from which test cases can be derived.

- Example:
  - Assume an application accepts integers in the range 100 – 499
  - Valid equivalence class partitioning is 100 to 499 inclusive
  - Invalid class partitioning would be numbers less than 100 and more than 499, decimal numbers, characters and strings.

Stubs

- Stubs are used during Top down integration testing.
- They are used in order to simulate the behaviour of the lower-modules that are not yet integrated.
- Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.
- Stubs are also used when the software needs to interact with an external system

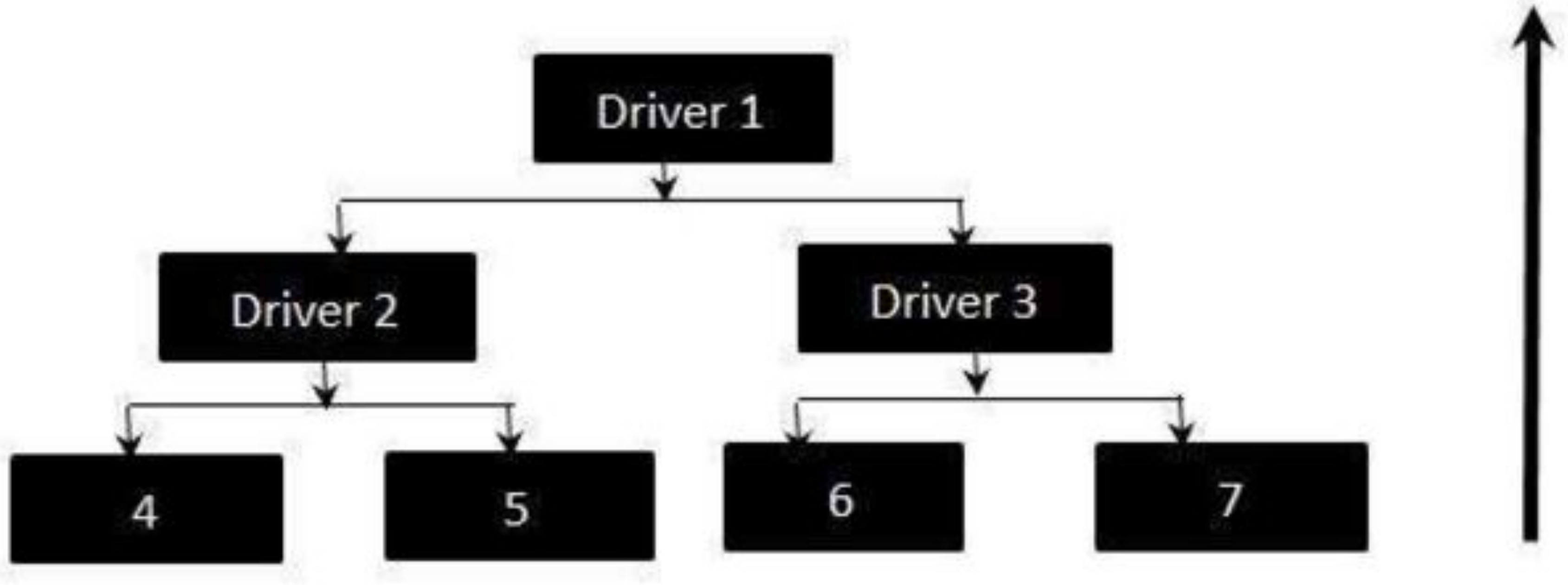
Example



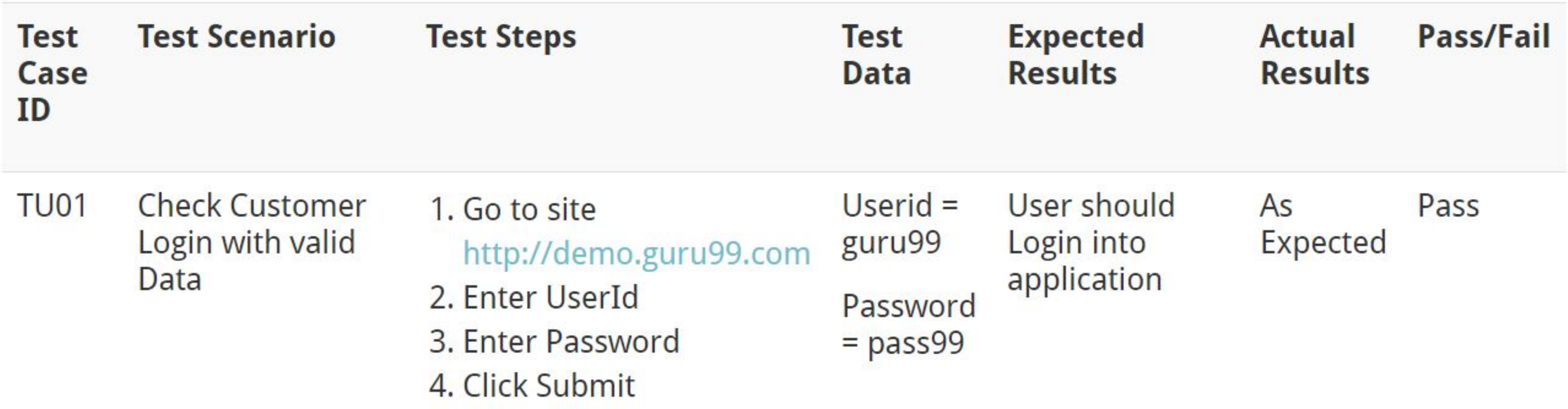
Drivers

- Drivers are the modules that act as temporary replacement for calling a module to give same output as the actual product.
- Used during bottom up testing in order to simulate the behaviour for upper level modules that are not yet integrated.
- They are also used when the software needs to interact with an external system.
- They are usually complex than stubs.

Example



How to write a test case



Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Customer Login with valid Data	1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit	Userid = guru99 Password = pass99	User should Login into application	As Expected	Pass

Software Test Plan

- A test plan is a document describing the scope, approach, objectives, resources, and schedule of a software testing effort.
- It identifies the items to be tested, items not be tested, who will do the testing, the test approach followed, what will be the pass/fail criteria, training needs for team, the testing schedule etc.
- A test plan basically has the following structure

Contents of a test plan

- Introduction
- References
- Test Strategy and Approach
- Test Criteria
- Test Deliverables
- Assumptions and Risks
- Responsibilities
- Resource Requirements
- Training Needs
- Defect logging and Tracking
- Metrics
- Approval Information
- Release Criteria